ESD-TR-77-141

VALIDATION OF THE PROTECTED DMS SPECIFICATIONS

I. P. Sharp Associates Limited
Ottawa, Canada

April 1977

Approved for Public Release;
Distribution Unlimited.

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

This technical report has been reviewed and is approved for publication.

William R Price

WILLIAM R. PRICE, Captain, USAF
Techniques Engineering Division

Roger R. Schell

ROGER R. SCHELL, Lt Colonel, USAF
ADP System Security Program Manager

FOR THE COMMANDER

Frank J. Emma

FRANK J. EMMA, Colonel, USAF
Director, Computer Systems Engineering
Deputy for Command & Management Systems

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-77-141	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) VALIDATION OF THE PROTECTED DMS SPECIFICATIONS		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gillian Kirkby Michael Grohn		8. CONTRACT OR GRANT NUMBER(s) F19628-76-C-0025
9. PERFORMING ORGANIZATION NAME AND ADDRESS I. P. Sharp Associates Limited Ottawa, Canada		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE 62702F Project 2801
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Command and Management Systems Electronic Systems Division Hanscom AFB, MA 01731		12. REPORT DATE April 1977
		13. NUMBER OF PAGES 124
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

mathematical, validation, Parnas, specifications, security, military, relational, data management, verification, certification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A mathematical validation of the formal specifications of the security-related functions of a secure (military sense) relational data management system is presented. The validation technique is described, and a sample set of Parnas-like specifications with all associated validation sheets are included.

Table of Contents

List of Tables

I INTRODUCTION

The purpose of this report is to prove mathematically that the specifications of the primitives of the protected DMS [1] correctly embody the protection principles designated by the mathematical model [2].

The validation technique is described, and the detailed validation of a representative sample of O-functions is included as an appendix.

II VALIDATION TECHNIQUE

2.1 Objective

The objective of the validation is to prove that the accesses of objects by subjects (in the secure DMS), which are formally specified in appendix IV of reference [1], do in fact conform to the axioms of the mathematical model.

These axioms are taken from section VI of reference [2], and are expressed using the notation and variables of the specifications. In addition, "$\succ_s$" means "security level dominance", and "$\succ_i$" means "integrity level dominance". Let S_v and I_v be the security and integrity levels, respectively, of variable V. M_v is the permission matrix associated with variable V. CUR_SEC and CUR_INT are the current security and integrity levels, respectively, of the subject of interest. Then the model axioms are:

(i) Direct Disclosure (Simple Security)

Subject observes V $\Rightarrow$ CUR_SEC $\succ_s$ S_v

(ii) Indirect Disclosure

Subject modifies V $\Rightarrow$ CUR_SEC $\prec_s$ S_v

(iii) Direct Modification

Subject modifies V $\Rightarrow$ CUR_INT $\succ_i$ I_v

(iv) Indirect "Contamination"

Subject observes V $\Rightarrow$ CUR_INT $\prec_i$ I_v

(v) Tranquility Principle

A) CUR_SEC and CUR_INT are constant for any subject; and

B) S_V and I_V are constant for any variable V.

(vi) Discretionary Access Control

Subject accesses V $\Rightarrow$ (Subject,access) ϵ M_V.

Essentially, the validation consists of proofs based on the inspection of specification statements, invariants, and case table results. Arguments of reasonableness are used in the determination of an O-function's cases, and to justify discretionary access control.

The mathematical notation and semantics used in the validation are those of the formal specifications, and are described in §B of appendix III of reference [1]. The flow of the validation is maintained by the use of clear English statements.

2.2 Invariants

Invariants are general "relations", or conditions involving variables of the specification which hold true between O-function invocations (i.e. between system states). They are proved inductively by showing that each O-function preserves each invariant [3]. The proofs involve inspection of the effects section of the relevant O-functions.

2.3 Case Tables

The purpose of the case tables is to identify the object accesses in all cases of the specified O-functions. Each case is defined by a distinct relationship among function inputs, and presents the protection levels of all variables observed and modified.

An inspection of a case table will prove that the protection level of each modified variable dominates the levels of all observed variables. Certain lemmas may be required to clearly justify the dominance relationship, and these are proved by invariants or the TRUE/FALSE value of exception conditions. In subsection 5.2 it is explained that this relationship ensures the satisfaction of non-discretionary requirements.

The set of all security-related O-functions is partitioned in such a way that each class of the partition consists of O-functions which possess analogous case tables. That is, input parameters, the number and type of accesses (observe or modify), and their levels are the same. Then the case tables of only certain representatives from each class are included in this report, in appendix I.

III DESIGN FUNDAMENTALS

The specification language was structured to facilitate the validation of the design. The semantics of the symbols used in the specifications most relevant to validation are summarized in this section.

3.1 Objects

Variables in the specifications represent the hidden kernel entities, directories, sign-on lists and the component entities of data base and user working area objects. Variables correspond to model objects since each possesses an identifier, a protection level, and a value.

The identifier of a variable is constructed in such a way as to indicate the characteristics of the design entity it represents, such as:

(i) user, kernel or data base environment (W, K or D);

(ii) the data type of the entity;

(iii) its owner (creator);

(iv) its name (for use in W);

(v) the kind of compound object of which it is a part (e.g. string, program, relation); and

(vi) its protection level.

The identifiers of variables isolated on a user basis (working area and kernel) do not include owner or level information, since they are "owned" by the current user, and their levels are stored in other variables. Their name is a mnemonic suggesting their role in the design.

3.2 Protection Level Assignment

Table 3.1 lists the protection levels which are assigned to the variables in the specifications.

The mathematical model explicitly gives the levels of the data base variables. Directories and sign-on lists are assigned the level of their contents. The level of a component entity of a data base object is taken to be the level component of its identifier.

Variables in the user's working area will assume the user's current (SIGNON) level. This is because they may be both observed and modified by the user.

In the hidden kernel area, certain variables contain data describing the current user, or his activities. Since the user set this data (by parameters) it will be assigned his current level. These variables are indicated in table 3.1 by level "K_CUR_LEVEL". The reserve table reflects successful reservation requests. The space quota gives the current session space resources of the user. The levels of the accumulator and temporaries reflect past "level" parameters. The current time is included to guarantee its correctness.

The levels of the accumulator and temporaries are set equal to the contents of K_LACC and K_LX (Y and Z too) respectively, since that is their purpose.

The open table is a five dimensional boolean array, which is partitioned according to the level dimension. That is, the level of each part of the open table is the level of the objects identified in that part, unless that level is strictly dominated by K_CUR_LEVEL. In that case the level of such a part is taken to be K_CUR_LEVEL [c.f. § 4.1].

Variable	Identifier	Level
Database (multi-level)		
directory	D_D(level)	level
sign-on list	D_Q(level)	level
exact size	D_E(o,n,t,l)	l
format	D_F(o,n,t,l)	l
history	D_H(o,n,t,l)	l
permission matrix	D_M(o,n,t,l)	l
open list	D_O(o,n,t,l)	l
reserve queue	D_R(o,n,t,l)	l
values	D_V(o,n,t,l)	l
maximum size	D_Z(o,n,t,l)	l
Kernel (isolated)		
current user level	K_CUR_LEVEL	K_CUR_LEVEL
current user identifier	K_CUR_ID	K_CUR_LEVEL
current time	K_CUR_TIME	K_CUR_LEVEL
session space quota	K_CUR_QTA	K_CUR_LEVEL
reserve table	K_RESERVE	K_CUR_LEVEL
open table	K_OPEN	K_CUR_LEVEL OR K_OPEN [level][1]
accumulator level	K_LACC	K_CUR_LEVEL
accumulator contents	K_IACC	K_LACC
accumulator format	K_FACC	K_LACC
accumulator values	K_VACC	K_LACC
level of temporary	K_LX	K_CUR_LEVEL
temporary contents	K_IX[2]	K_LX
temporary format	K_FX[2]	K_LX
temporary values	K_VX[2]	K_LX
Working area (isolated)		
return code	W_CODE	K_CUR_LEVEL
relation value table	W_Vname	K_CUR_LEVEL
relation format	W_Fname	K_CUR_LEVEL

Table 3.1 Protection Level Assignment

[1] The dominating level is assigned to each part of the open table.

[2] Y and Z temporaries as well.

3.3 Object Access

The accessing of objects is represented in the specifications by usage of a variable's identifier. An observe access is represented by a variable's appearance in a V-function derivation, an exception condition, or on the right-hand side of the "arrow" (←) in an assignment statement in an effects section. A modification access is represented by the appearance of a variable on the left-hand side of the "arrow" in an assignment statement.

3.4 Subjects

A subject is a process associated with each O-function invocation, whose sole purpose is to have the capability (authorization) to perform the observations and modifications required by the effects of the O-function.

When an O-function is invoked, one subject sets the return code (W_CODE) and the levels of the accumulator (K_LACC) and temporaries (K_LX, K_LY and K_LZ). The level of this subject is K_CUR_LEVEL [c.f. table 3.1]. Another subject performs all *other* modifications. An inspection of the case tables in appendix I will reveal that these modifications are all either at level "lv" (a parameter) or level "LA" (level of the accumulator), except for SIGNON, SIGNOFF and MOVE. Therefore the level of the second subject is established as that level.

Since the three exceptions are functions which perform multi-level observations and modifications, they are restricted to be performed by "trusted" processes such as the user control process (UCP) and the data base administrator (DBA).

IV PROOF OF INVARIANTS

4.1 Minimum K-level

4.1.1 Invariant

K_LACC $\succeq$ K_CUR_LEVEL

and

K_Lx $\succeq$ K_CUR_LEVEL, for x = X,Y,Z.

The levels of the kernel accumulator and temporaries always dominate the current user's level (K_CUR_LEVEL).

4.1.2 Requirement

This invariant ensures that parameter data (which is at level K_CUR_LEVEL) is never found in a kernel entity at a strictly dominated level (prohibits "write-downs").

4.1.3 Proof

An inspection of the case tables in appendix I will reveal that K_LACC is set by the following primitives:

(i) DKD, DKQ, LIST_DOWN, and WKB;

An inspection of their specifications will reveal that the level K_LACC effect is:

K_LACC ← K_CUR_LEVEL

(ii) DKE, DKH, DKM, DKR, DKV, and DKZ;

An inspection of their specifications will reveal that the level K_LACC effect is:

K_LACC ← LEV IF LEV ≻ K_CUR_LEVEL ELSE K_CUR_LEVEL

where LEV is the level parameter

Clearly, these assignments result in the relation:

K_LACC ≻ K_CUR_LEVEL

(iii) ASSIGN

The level effect statement is:

K_LACC ← K_Lsource IF source ∈ {ACC,X,Y,Z}.

An inspection of the case tables will reveal that <u>only</u> the ASSIGN primitive modifies the temporaries X, Y and Z. The function of ASSIGN is such that data can be assigned to temporaries from two sources only:

- the kernel accumulator; or
- a working area (W) value.

In the first case:

K_Lx ≻ K_CUR_LEVEL, x ∈ {X,Y,Z},

by the conclusions of (i) and (ii), since K_Lx was in K_LACC once.

In the second case, K_Lx is not modified, so the invariant is not affected. Therefore, K_LACC ≻ K_CUR_LEVEL if source ∈ {X,Y,Z}.

If source = ACC, there is no change in K_LACC and the invariant is unaffected.

If source ∉ {ACC,X,Y,Z}, then K_LACC is not changed, so the invariant is unaffected.

Note that the assignment of a working area value will occur only when the data conforms to the format data in the kernel, prohibiting assignment of values only to null kernel entities. Therefore, the levels of the kernel entities will dominate the values by the above arguments.

(iv) SIGNOFF

The effect statements:

$K\_LACC, K\_Lx \leftarrow \emptyset \quad , x \in \{X,Y,Z\}$
$K\_CUR\_LEVEL \leftarrow \emptyset$

trivially satisfy the invariant.

Q.E.D.

4.2 Open Table

4.2.1 Invariant

Subscript a variable by t to indicate its value in the system state at time t. Then:

$$\text{K\_OPEN}_t(\text{owner},\text{name},\text{type},\text{level},\text{access}) = \text{TRUE}$$

$$\Longrightarrow$$

$$\text{K\_OPEN}_{t-1}(\text{owner},\text{name},\text{type},\text{level},\text{access}) = \text{TRUE}$$

$$\text{OR}$$

$$(\text{K\_CUR\_ID},\text{access}) \in \text{D\_M}_{t-1}(\text{owner},\text{name},\text{type},\text{level})$$

$$\text{OR}$$

$$\text{K\_CUR\_ID} = \text{owner}$$

If an object has some access granted to it in a user's open table, then it was there in the preceeding system state, or it was authorized in the object's permission matrix in the preceeding state (when OPEN was requested).

4.2.2 Requirement

This invariant assures that the discretionary authorization mechanisms function correctly.

4.2.3 Proof

An inspection of the case tables in appendix I reveals that only O_APPEND, O_DELETE and SIGNOFF modify K_OPEN. Effects statements in O_DELETE and SIGNOFF set the K_OPEN entry of interest to FALSE, so only O_APPEND is relevant to this invariant.

There are two possibilities:

(i) O_APPEND was invoked at time (t-1), producing the system state at time t;

or (ii) O_APPEND was not invoked at time (t-1).

Suppose O_APPEND was not invoked at time (t-1). If K_OPEN(owner,name,type,level,access) = TRUE at time t, then it must be TRUE at time (t-1) as well, since K_OPEN is not modified in this case.

Otherwise, suppose O_APPEND was invoked at time (t-1). If the open table entry of interest is TRUE at time (t-1), then O_APPEND will return exception code "DO", and the entry will remain unchanged to time t.

If the entry is FALSE at time (t-1) and TRUE at time t, then exception ND = FALSE. That is, by the derivation of the ND exception:

$$\sim(\mathrm{K\_CUR\_ID} \neq \mathrm{owner}) \wedge (\nexists x(\mathrm{K\_CUR\_ID},x) \in \mathrm{D\_M}_{t-1}(\mathrm{id}))$$

$$\equiv$$

$$(\mathrm{K\_CUR\_ID} = \mathrm{owner}) \vee (\exists x(\mathrm{K\_CUR\_ID},x) \in \mathrm{D\_M}_{t-1}(\mathrm{id}))$$

Effect [1] of O_APPEND states:

K_OPEN(Access_set_O(id))= TRUE

Access_set_O returns the set of tuples S:

$$S = \{(id,x) : (OWN = K\_CUR\_ID) \vee (K\_CUR\_ID,x) \in D\_M_{t-1}(id)\}$$

by the derivation of the Access_set_O and Aith_O V-functions. The "access" in the invariant is one of these x's, since these are the accesses authorized.

Q.E.D.

4.3 Identification

4.3.1 Invariant

$$D\_V(id) \in K\_Vx \quad \text{and} \quad D\_F(id) \in K\_Fx$$

$$\Longrightarrow$$

$$K\_Ix = id, \text{ for } x = ACC,X,Y,Z.$$

At all times the identifiers (i.e. K_IACC, K_IX, K_IY, K_IZ) of the contents of the kernel accumulator and temporaries are a correct indication of the identity of their contents.

4.3.2 Requirement

This invariant is required for discretionary authorization, to ensure that data cannot masquerade when being presented to a useı (KWA) or copied to the data base (KD...).

4.3.3 Proof

An inspection of the case tables in appendix I will reveal that K_IACC is modified by the O-functions in table 4.1. An inspection of the K_IACC effect in each function specification (included in table 4.1) will reveal that it is appropriate for the function.

Since only ASSIGN affects the temporaries, they contain only data previously in the accumulator. Assigning a user working area value to the values component of the accumulator or a temporary does not affect its identity.

Q.E.D.

Primitive	Identification Effect
DKD	K_IACC ← (K_CUR_ID,'D','R',lv,'V')
DKQ	K_IACC ← (K_CUR_ID,'Q','R',lv,'V')
LIST_DOWN	K_IACC ← (K_CUR_ID,DEF_NAME,'R',K_CUR_LEVEL,'V')
DKC	K_IACC ← (id,C) , C $\in$ {E,H,M,R,V,Z}
ASSIGN	K_Itarget ← K_Isource IF source $\in$ {ACC,X,Y,Z}
WKB	K_IACC ← (K_CUR_ID,n,'S',K_CUR_LEVEL,'V')
SIGNOFF	K_IACC ← $\emptyset$

Table 4.1 Table of Identification Data

V JUSTIFICATION OF PROTECTION

5.1 Non-discretionary Policy

The secure DMS has been designed in such a way that there is no modification access without an observation access, and vice versa.

An O-function execution involves two subjects: One executes at the single level of the data being modified, and performs all effects except the setting of W_CODE, K_LACC, K_LX, K_LY and K_LZ. (This is "Subject 2" in appendix I.) The other subject executes at the user's current level (K_CUR_LEVEL), and sets them ("Subject 1").

Therefore, the first four (non-discretionary) model axioms [c.f. § 2.1] are maintained by the specifications if it can be proved that for each O-function, the level of the modified variables dominates the levels of all observed variables, for each subject. This follows from the definition of protection levels and protection dominance [2].

The representative sample of case tables in appendix I do indeed prove that this is true for every O-function except SIGNON, SIGNOFF and MOVE.

Since the three exceptions violate the axioms, each must be executed by a special process, "trusted" to perform its function correctly. The user control process (UCP) executes SIGNON and SIGNOFF in response to a human user's request. MOVE may be invoked only by the data base administrator's process.

These "trusted" processes are required to execute at the system-high protection level, and therefore maintain at least the simple security [c.f. § 2.1] and indirect "contamination" axioms.

5.2 Tranquility Principle

It is assumed that the protection levels of the subjects performing the O-function effects remain constant.

Variables in the user working area are tranquil for the following reasons:

(i) they are completely isolated from other users; and

(ii) their level, K_CUR_LEVEL (by definition), is modified only by SIGNON (initialized) and SIGNOFF (purged).

In the data base, the protection level of an object is a parameter of the access (by the structure of identifiers [c.f. § 3.1]). Each different level parameter indicates a different object is to be accessed. Additionally, the design allows no movement of identifiers from one directory to another.

Isolation of the kernel working area, and the constancy of K_CUR_LEVEL ensure that the following kernel entities maintain a constant protection level:

K_CUR_LEVEL, K_CUR_ID, K_CUR_QTA,

K_CUR_TIME, K_OPEN, K_RESERVE,

K_LACC, K_LX, K_LY, K_LZ.

However, the level of the contents of the accumulator or a temporary may be changed by various O-functions. This apparent violation of tranquility is acceptable for the following reasons:

(i) an inspection of the case tables in appendix I will reveal that every modification of K_LACC, K_LX, K_LY or K_LZ has an associated modification of K_VACC, K_VX, K_VY or K_VZ, respectively; and

(ii) this modification is specified by means of an assignment statement, which requires the previous contents of the accumulator to be purged upon re-assignment (by definition).

In conclusion, it is evident that the tranquility principle of the model is maintained.

5.3 Discretionary Authorization

Discretionary authorization policy requires explicit permission to have been extended before a user may access a data base object.

The primitive function which establishes such permission is KDM. An inspection of its specification [1] will prove its correctness. That is, the "NO" exception ensures that there is discretionary authorization to extend discretionary authorization. This is made possible by defining ownership of a variable (indicated in its identifier) to imply complete discretionary authorization. The "IC" and "IV" exceptions ensure that the accumulator contains the appropriate permission matrix.

The only means of transferring data base objects to the user is by means of the KWA primitive, and the Discretionary_kwa V-function checks for appropriate authorization. The Open Table [c.f. § 4.2] and Identification [c.f. § 4.3] invariants ensure that the mechanisms involved in this check function correctly.

The only means of storing data in the data base is by means of the WDV, KDM and KDV O-functions. The NO exception in each of these functions checks for discretionary authorization.

The Open Table and Identification invariants ensure that all data in the hidden kernel area is correctly subject to discretionary authorization.

Table 5.1 summarizes the discretionary authorization mechanisms in the O-functions, leading to the conclusion that the discretionary authorization model axiom is maintained.

Primitive	Discretionary Authorization	Primitive	Discretionary Authorization
1. APP_DIR	directory is exempt	DKM	NO exception
2. DEL_DIR	directory is exempt	DKQ	signon list is exempt
3. REP_DIR	directory is exempt	DKR	NO exception
4. INIT	ownership	DKV	NO exception
5. DESTROY	ownership	DKZ	NO exception
6. RES	NO exception	WKB	ownership
7. REQ	NO exception	KDM	NO exception
8. REL	RS exception	KDV	NO exception
9. SIGNON	trusted process	KDZ	ownership
10. SIGNOFF	trusted process	WDV	NO exception
11. O_APPEND	ND exception	KWA	Discretionary_kwa
12. O_DELETE	NO exception	PROJECTW	ownership
13. APPEND	hidden	SELECTW	ownership
14. ASSIGN	hidden	APPENDW	ownership
15. CONCAT	hidden	CROSS	ownership
16. EXTRACT	hidden	ARITH	ownership
17. SELECT	hidden	ASSIGNW	ownership
18. PROJECT	hidden	SIZE	ownership
19. LIST_DOWN	exemption	APFOR	ownership
20. DKD	directory is exempt	DIFF	ownership
21. DKE	NO exception	MOVE	directory is exempt
22. DKH	NO exception		

Table 5.1 Discretionary Authorization Relevant to Each O-function

APPENDIX I

O-FUNCTION CASES

Table of Contents

A.1.1 Introduction

This appendix contains detailed case tables [c.f. § 2.3] for certain O-function specifications. For each such O-function the following is included:

(i) its formal specification from appendix IV of reference [1];

(ii) flow diagrams illustrating the cases for each of the two subjects [c.f. § 3.4] performing the O-function; and

(iii) its set of case tables. The purpose of each case table is to prove that the level of each modified variable dominates the levels of all variables observed in that case.

The security related primitives are the only ones requiring certification, and these may be categorized according to analogous case tables. The case tables for a representative sample of O-functions is included in this appendix. The sample consists of at least one O-function from each category (subsection A.1.3).

Three primitive functions break some of the rules of the "strict" protection policy, but they are essential for computerized data management. These are SIGNON, SIGNOFF and MOVE, and their case tables are found in subsection A.1.4.

A.1.2 Notation Used in Case Tables

The symbols and mnemonics used in the case tables are taken from the formal specifications, and are described in appendicies III and IV of reference [1].

Additionally, the following abbreviations are used:

abbreviation	meaning
W	- User's current protection level (K_CUR_LEVEL)
LA	- Level of the kernel accumulator
LX	- Level of a kernel temporary
$L_1 \lceil L_2$	- The dominant level of L_1 and L_2
$L_1 \lfloor L_2$	- The dominated level of L_1 and L_2
id[LEV]	- The level component of the identifier
LEV	- An abbreviation for id[LEV]
ACC	- Kernel accumulator
UCP	- User Controller Process

A.1.3 The Sample Validations

The set of all security-related O-functions are categorized below according to analogous case tables. Those whose case tables are found in this section are so indicated.

	Category	Case Tables Included	Not Included
1.	Directory manipulation	APP_DIR	DEL_DIR, REP_DIR
2.	Object existence	INIT	DESTROY
3.	Object reservation	REQ	RES, REL
4.	Access authorization	O_APPEND	O_DELETE
5.	Accumulator manipulation	APPEND, CONCAT	ASSIGN, EXTRACT, SELECT, PROJECT, LIST_DOWN
6.	Transfer to accumulator	DKD, DKE	DKH, DKM, DKQ, DKR, DKV, DKZ
7.	Transfer to data base	KDM, KDV, KDZ	WDV
8.	Working area	KWA	WKB
9.	Access DMS	SIGNON, SIGNOFF	
10.	Data base administrator	MOVE	

Table A.1.1 Table of O-function Categories

A.1.3.1 APP_DIR

```
(A)  O-function  APP_DIR(lv,n,t,lz)

          * Append a tuple to a data base directory, where:             *
          *      (i)  lz = 0 if it's an object's defining entry;         *
          *     (ii)  lz ≠ 0 if it's an object registration at "lv".     *

     parameter types

          level lv, name n, type t, level_zero lz

     exception             * Check legality of parameter lz first. *

          IR:[1] FALSE IF lz = 0  ELSE (lz = lv) v (lz ⊁ lv)        * Zero or strictly dominating *
          IL:[2] lv ⊁ K_CUR_LEVEL                                   * Append must be a write-up   *
         *DD:  (K_CUR_ID,n,t,*) ∈ ⊢D_D(lv)⊣                         * Entry is there already      *

     effect       * Note that effect [2] gives the semantics of "*DD". *

          [1]  D_D(lv) ← ⊢ ⊣ ∪ (K_CUR_ID ,n,t,lz)       * Append the entry     *
         *[2]  W_CODE ← DN   IF  K_CUR_LEVEL = lv       * Avoid a "write-down" *
```

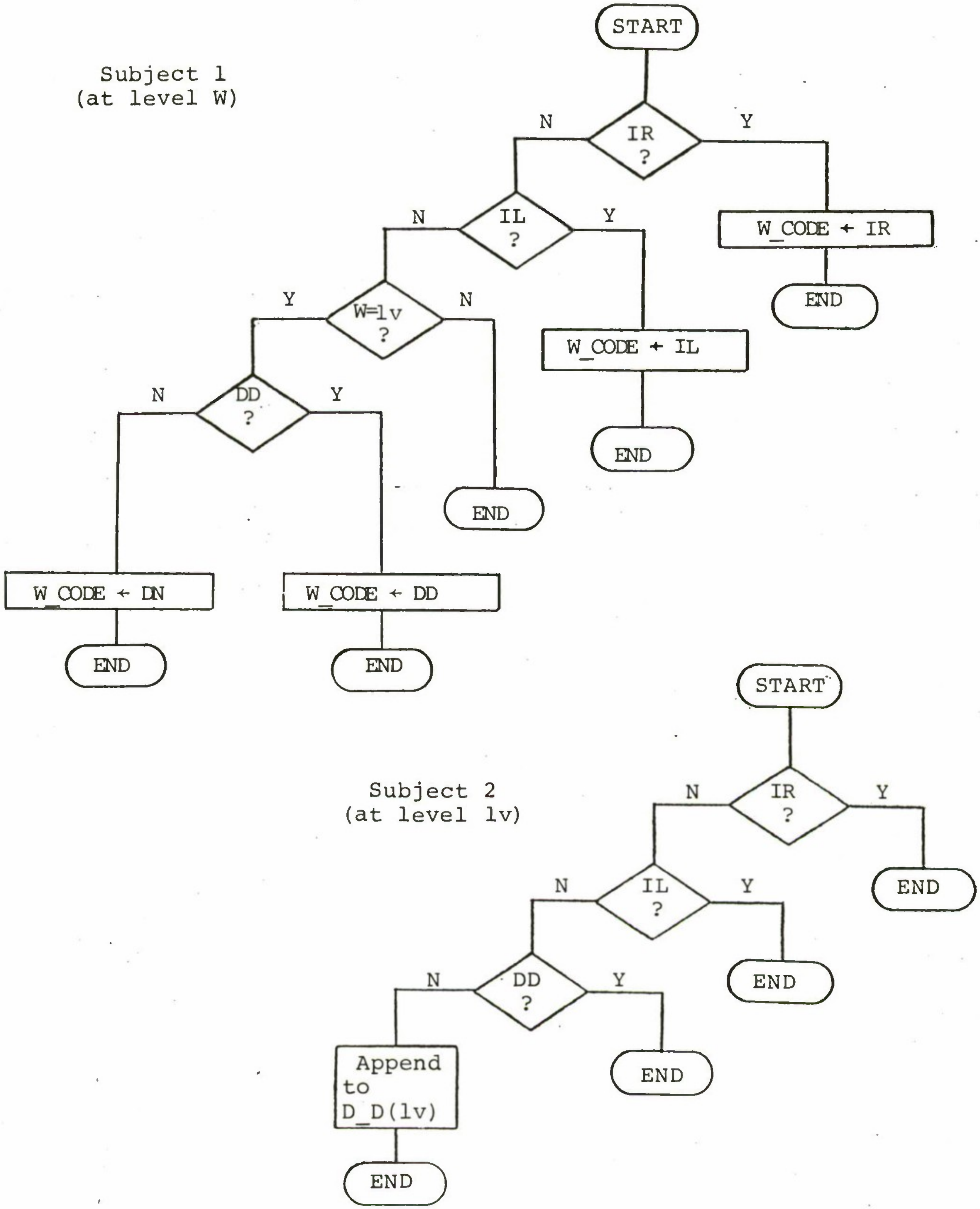
(B) Access table

Variables Observed	Variables Modified	Variables Observed and Modified
K_CUR_ID K_CUR_LEVEL	W_CODE	D_D(lv)

[1] Exception statements containing "IF" are of the form: E_1 IF E_2 ELSE E_3, where E_i are logical expressions with TRUE/FALSE values. The semantics of such a statement is: exception is value of E_1 if E_2 evaluates to TRUE, else exception is value of E_3.

[2] The asterisk by DD indicates that code DD is returned ONLY if K_CUR_LEVEL = lv. The asterisk in the directory tuple indicates that any value found in the LEVEL domain will satisfy the membership condition.

APP_DIR



Subject 1
(at level W)

Subject 2
(at level lv)

PRIMITIVE: APP_DIR CASE: 1 SUBJECT: 1

CONDITIONS: IR
Register attempted at level not strictly dominated by that of the definition entry.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	lv, n, t, lz	W	W_CODE	W
CONSTANTS:	O, IR	Unclass		
VARIABLES:				
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APP_DIR CASE: 2 SUBJECT: 1

CONDITIONS: $(\sim IR) \wedge IL$
Directory level does not strictly dominate the user's current level.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	lv, n, t, lz	W	W_CODE	W
CONSTANTS:	O, IL	Unclass		
VARIABLES:	K_CUR_LEVEL			
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APP_DIR CASE: 3 SUBJECT: 1

CONDITIONS: (~IR) ∧ (~IL) ∧ (W=lv) ∧ DD
The directory entry already exists.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	lv, n, t, lz	W	W_CODE	W
CONSTANTS:	O, DD	Unclass		
VARIABLES:	K_CUR_LEVEL D_D(lv) K_CUR_ID	W lv=W W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APP_DIR CASE: 4 SUBJECT: 1

CONDITIONS: (~IR) ∧ (~IL) ∧ (W=lv) ∧ (~DD)
No exceptions.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	lv, n, t, lz	W	W_CODE	W
CONSTANTS:	O, DN	Unclass		
VARIABLES:	K_CUR_LEVEL D_D(lv) K_CUR_ID	W lv=W W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APP_DIR CASE 1 SUBJECT: 2

CONDITIONS: (~IR) ∧ (~IL) ∧ (~DD)
No exceptions

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	lv, n, t, lz	W	D_D (lv)	lv
CONSTANTS:	O, DN	Unclass		
VARIABLES:	K_CUR_LEVEL K_CUR_ID D_D (lv)	W W lv		
HIGHEST LEVEL OBSERVED:		lv	LOWEST LEVEL MODIFIED:	lv

LEMMA: lv ⊁ W
PROOF: ~IL

A.1.3.2 INIT

(A) O-function INIT(n,t,lv,s)

```
     * Initialize all component entities of an object.  *
     * If parameter "size" is negative, and the session *
     * quota is less than the absolute value of the     *
     * size, then the maximum size of the object is set *
     * equal to the current session quota.              *
```

parameter types

```
     name n, type t, level lv, size s
```

abbreviation

```
     id = K_CUR_ID,n,t,lv                    * Identifier of new object *
```

exception

```
     IL:  lv ⊁ K_CUR_LEVEL                   * Illegal level for modification *
    *DE:  (K_CUR_ID,n,t,ZERO) ∉ D_D(lv)      * Directory entry is required    *
    *DD:  ⊢D_E(id)⊣ ≠ ∅                      * It exists already              *
    *SZ:  s >  K_CUR_QTA                     * Size is too large              *
                        * Note:  negative size case is included here          *
```

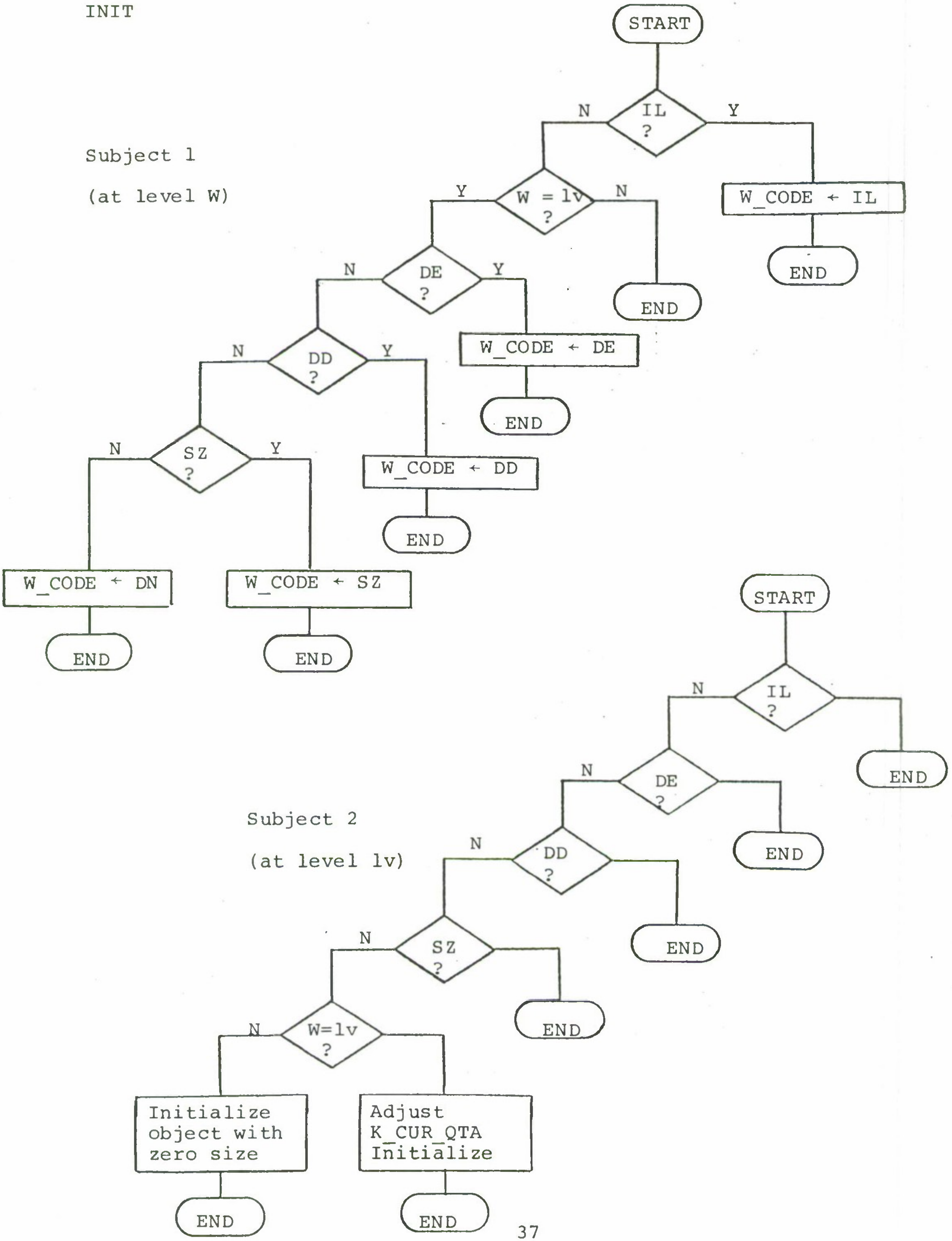
effect

```
     [1]  D_E(id) ← ZERO                     * Initialize current size        *
     [2]  D_H(id) ← (K_CUR_TIME,K_CUR_ID,K_CUR_TIME)    * Initialize object history *
     [3]  D_Z(id) ← Size_init(s) IF  K_CUR_LEVEL = lv  ELSE  ZERO   * Use QUOTA only at *
     [4]  K_CUR_QTA ← ⊢ ⊣ - Size_init(s)  IF  K_CUR_LEVEL = lv      * user's current level *
    *[5]  W_CODE ← DN  IF  K_CUR_LEVEL = lv
```

(B) Access table

Variables Observed	Variables Modified	Variables Observed and Modified
	D_Z(id)	D_E(id)
K_CUR_LEVEL	D_H(id)	K_CUR_QTA
D_D(lv)	W_CODE	
K_CUR_ID		
K_CUR_TIME		

INIT



Subject 1
(at level W)
START
IL ?
Y
W_CODE ← IL
END
N
W = lv ?
N
END
Y
DE ?
Y
W_CODE ← DE
END
N
DD ?
Y
W_CODE ← DD
END
N
SZ ?
Y
W_CODE ← SZ
END
N
W_CODE ← DN
END
Subject 2
(at level lv)
START
IL ?
END
N
DE ?
END
N
DD ?
END
N
SZ ?
END
N
W=lv ?
Adjust K_CUR_QTA Initialize
END
N
Initialize object with zero size
END

PRIMITIVE: INIT CASE: 1 SUBJECT: 1

CONDITIONS: IL
Level of object to be initialized does not
dominate user's current level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n,t,lv,s	W	W_CODE	W
CONSTANTS: IL	Unclass		
VARIABLES: K_CUR_LEVEL	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: INIT CASE: 2 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W=lv) ∧ DE
Object is not defined in the directory, and
the object level equals the user's current level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n,t,lv,s	W	W_CODE	W
CONSTANTS: ZERO, DE	Unclass		
VARIABLES: K_CUR_LEVEL K_CUR_ID D_D(lv)	 W W lv=W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: INIT CASE: 3 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W=lv) ∧ (~DE) ∧ DD
The object has been previously initialized

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n,t,lv,s	W	W_CODE	W
CONSTANTS: ZERO, DD	Unclass		
VARIABLES: K_CUR_LEVEL K_CUR_ID D_D(lv) D-E(id)	 W W lv=W lv=W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: INIT CASE: 4 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W=lv) ∧ (~DE) ∧ (~DD) ∧ SZ
The requested size exceeds user's current quota

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n,t,lv,s	W	W_CODE	W
CONSTANTS: ZERO, SZ	Unclass		
VARIABLES: K_CUR_LEVEL K_CUR_ID D_D(lv) D_E(id) K_CUR_QTA	W W lv=W lv=W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: INIT CASE: 5 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W=lv) ∧ (~DE) ∧ (~DD) ∧ (~SZ)
No exceptions.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	n,t,lv,s	W	W_CODE	W
CONSTANTS:	ZERO, Ø, DN	Unclass		
VARIABLES:	K_CUR_LEVEL K_CUR_ID D_D(lv) D_E(id) K_CUR_QTA K_CUR_TIME	W W lv=W lv=W W W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: INIT CASE: 6 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W≠lv)
The object is initialized at a strictly dominating level.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	n,t,lv,s	W		W[3]
CONSTANTS:		Unclass		
VARIABLES:	K_CUR_LEVEL	W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

[3] Note that this is the "null" modification, that is, W_CODE is set in such a way that it signals that "lv strictly dominates K_CUR_LEVEL", by ~IL.

PRIMITIVE: INIT CASE: 1 SUBJECT: 2

CONDITIONS: (~IL) ∧ (~DE) ∧ (~DD) ∧ (~SZ) ∧ (W=lv)
No exceptions. Initialize object at user's current level.

	OBSERVED	LEVEL	MODIFIED	LEVEL

PARAMETERS:	n,t,lv,s	W	D_E(id)
			D_H(id)
			D_Z(id)
			K_CUR_QTA
				lv=W
				lv=W
				lv=W
				W

CONSTANTS:	ZERO, Ø, DN	Unclass		
VARIABLES:	K_CUR_LEVEL	W		
	K_CUR_ID	W		
	D_D(lv)	lv=W		
	D_E(id)	lv=W		
	K_CUR_QTA	W		
	K_CUR_TIME	W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: INIT CASE: 2 SUBJECT: 2

CONDITIONS: (~IL) ∧ (~DE) ∧ (~DD) ∧ (~SZ) ∧ (W≠lv)
No exceptions. Initialize object at level strictly dominating user's current level, giving it zero size.

	OBSERVED	LEVEL	MODIFIED	LEVEL

PARAMETERS:	n,t,lv,s	W	D_E(id)
			D_H(id)
			D_Z(id)
				lv
				lv
				lv

CONSTANTS:	ZERO, Ø, DN	Unclass		
VARIABLES:	K_CUR_LEVEL	W		
	K_CUR_ID	W		
	D_D(lv)	lv		
	D_E(lv)	lv		
	K_CUR_QTA			
	K_CUR_TIME	W		
HIGHEST LEVEL OBSERVED:		lv	LOWEST LEVEL MODIFIED:	lv

Lemma: lv ≻ W
Proof: ~IL

A.1.3.3

```
(A)  O-function  REQ(o,n,t)

          * Reserve an object (at the current level) if it's *
          * available.  If not, wait until it is available   *

     parameter types

          user o, name n, type t

     abbreviation

          id = o,n,t,K_CUR_LEVEL                  * Identifier of object to reserve *

     exception

          NO:  K_OPEN(id,RSRV)[4] = FALSE         * The object has not been opened *
          RS:  id̄ ∈ ⊢K_RESERVE⊣                   * I've already reserved it       *
          DL:  (⊢D_R(id̄)⊣ ≠ ∅) ∧ (⊢K_RESERVE⊣ ≠ ∅)  * Dead-lock hazard               *
               * If current process must wait while holding objects, dead-lock could result. *

     WAIT_UNTIL(⊢D_R(id)⊣ =∅)                     * Wait until object is not reserved. *
          * This specification statement is to be performed AFTER *
          * all exceptions have been tested for, and BEFORE the   *
          * effects take place.                                   *

     effect

          [1]  D_R(id) ← K_CUR_ID                 * Reserve the object             *
          [2]  K_RESERVE ←⊢ ⊣∪ (id)               * Append an entry to reserve table *
          [3]  W_CODE ← DN
```

(B) Access table

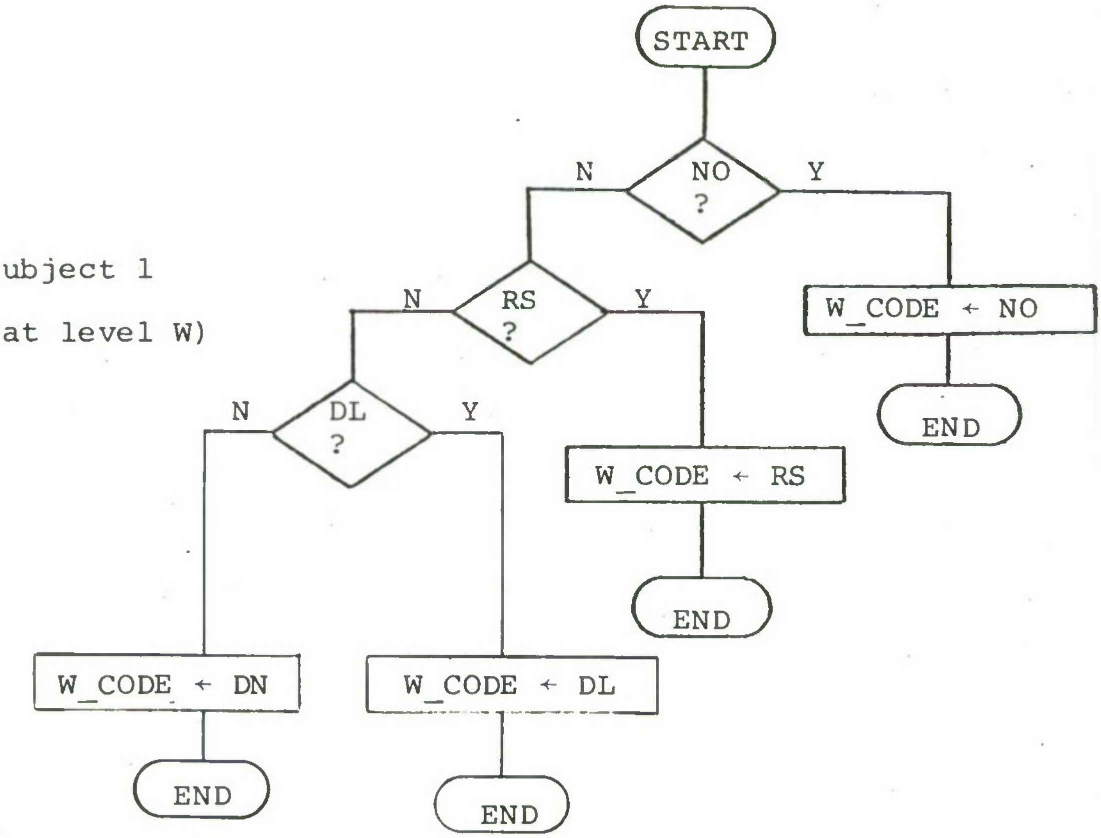
Variables Observed	Variables Modified	Variables Observed and Modified
K_OPEN K_CUR_LEVEL K_CUR_ID	W_CODE	K_RESERVE D_R(id)

[4] The open table is a five-dimensional array, with the "level" dimension assuming a lattice structure.

REQ

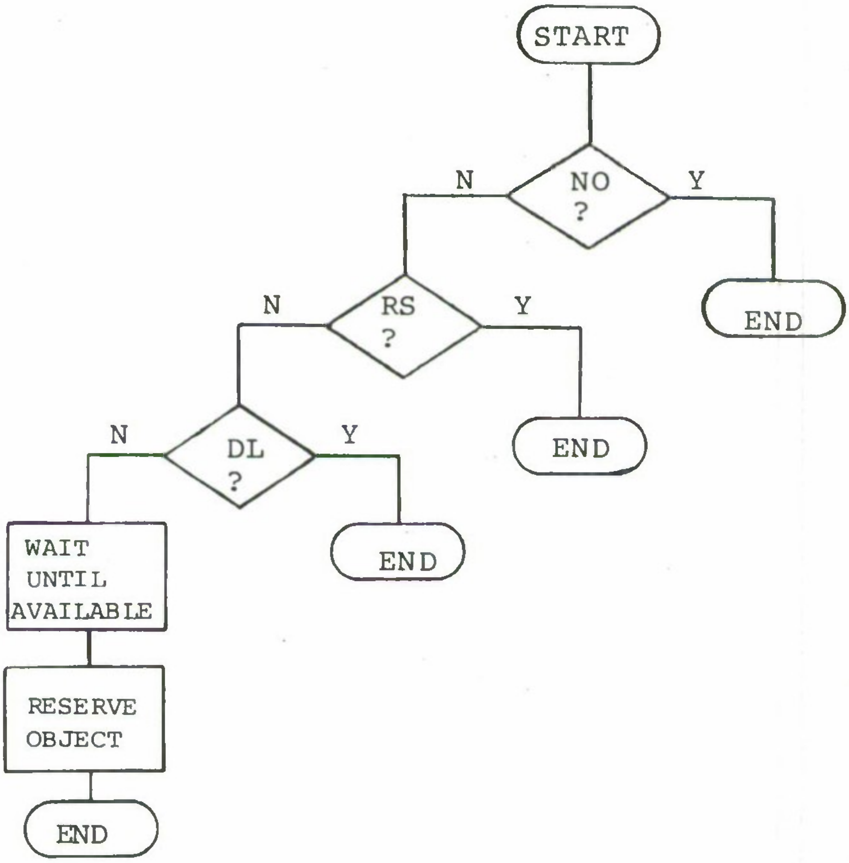
Subject 1

(at level W)



Subject 2

(at level W)



PRIMITIVE: REQ CASE: 1 SUBJECT: 1

CONDITIONS: NO
Object is not open.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: o,n,t	W	W_CODE	W
CONSTANTS: RSRV, NO	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN	W W[5]		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: REQ CASE: 2 SUBJECT: 1

CONDITIONS: (~NO) ∧ RS
User already has object reserved.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: o,n,t	W	W_CODE	W
CONSTANTS: RSRV, RS	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN K_RESERVE	W W[5] W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

[5] Note that reservation is restricted to a user's current level.

PRIMITIVE: REQ CASE: 3 SUBJECT: 1

CONDITIONS: (~NO) ∧ (~RS) ∧ DL
Object is reserved by another user, and to queue for it would risk deadlock.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: o,n,t	W	W_CODE	W
CONSTANTS: RSRV, Ø, DL	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN K_RESERVE D_R(id)	W W W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: REQ CASE: 4 SUBJECT: 1

CONDITIONS: (~NO) ∧ (~RS) ∧ (~DL)
No exceptions

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: o,n,t	W	W_CODE	W
CONSTANTS: RSRV, Ø, DN	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN K_RESERVE D_R(id)	W W W W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: REQ CASE 1 SUBJECT: 2

CONDITIONS: (~NO) ∧ (~RS) ∧ (~DL)
No exceptions. Reserve object on behalf of user.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: o,n,t	W	K_RESERVE	W
CONSTANTS: RSRV, Ø, DN	Unclass	D_R(id)	W
VARIABLES:			
K_CUR_LEVEL	W		
K_OPEN	W		
K_RESERVE	W		
D_R(id)	W		
K_CUR_ID	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

A.1.3.4   O_APPEND

```
(A)   O-function  O_APPEND(id)

          * This primitive opens an object for all possible access by W, *
          * by appending a tuple to K_OPEN for every authorized access.   *
          * An open is required to:  (i)  allow object access; and        *
          *       (ii)  prohibit the purging of an accessed object.       *

      parameter types

          identifier id (OWN,NAM,TYP,LEV)[6]

      exception

          IL:   (K_CUR_LEVEL ⊁ LEV)[7] ∧ (LEV ⊁ K_CUR_LEVEL)   * Incomparable level     *
         *DO:   K_OPEN(id,*) = TRUE                           * Object is open already *
         *NE:   D_E(id) = ∅                                   * Object does not exist  *
         *ND:   (K_CUR_ID ≠ OWN) ∧ ((K_CUR_ID,*) ∉ D_M(id))
                * There is no discretionary authorization whatsoever.                 *

      effect

          [1]   K_OPEN(Access_set_O(id))= TRUE           * Set each authorized access to TRUE *
          [2]   D_O(id)← ⊢ ⊣ ,K_CUR_ID  IF  Opened_O(id) ∧ (LEV ≻ K_CUR_LEVEL)
         *[3]   W_CODE ← DN  IF  Opened_O(id) ∧ (K_CUR_LEVEL ≻ LEV)  ELSE  ND  IF K_CUR_LEVEL ≻ LEV
                * Note that the open list, D_O(id), is NOT modified for strictly dominated objects. *
```

(B)   Access table

Variables Observed	Variables Modified	Variables Observed and Modified

K_CUR_LEVEL
K_CUR_ID
D_E(id)
D_M(id)
	W_CODE	K_OPEN
		D_O(id)

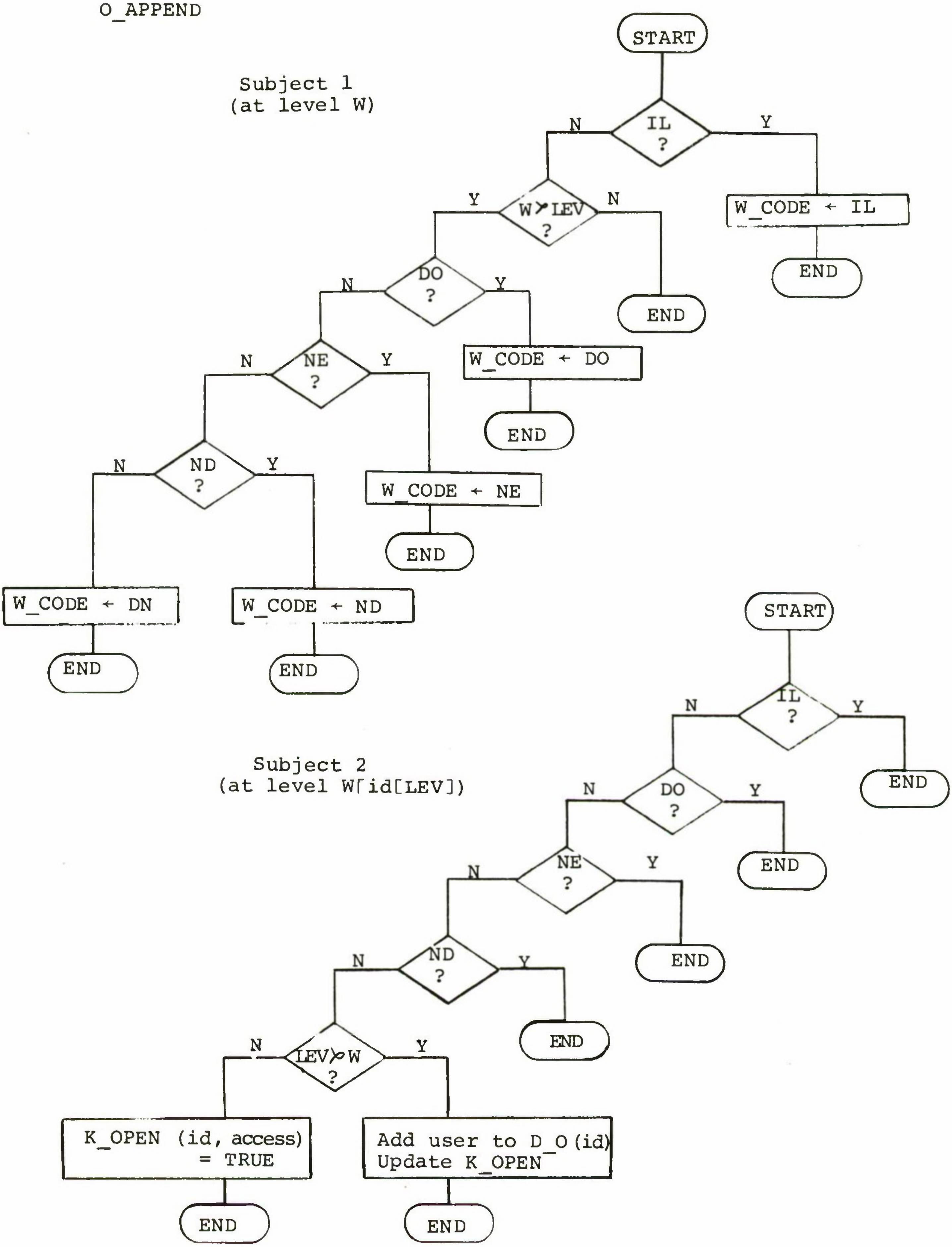
[6] Type "id" has previously been defined, so "(OWN,NAM,TYP,LEV)" should appear in a comment, instead of in the parameter declaration.

[7] "LEV" is an abbreviation for "id[LEV]".

```
(C)  O_APPEND  V-functions

(i)  V-function  Access_set_O(id) : open

          * Return a set of open table tuples; one for each authorized access mode. *

     range

          ZERO to NUM_MODES of (identifier,access)

     parameters

          identifier  id  (OWN,NAM,TYP,LEV)

     derivation

           (id,APCY)  IF  Auth_O(LEV,K_CUR_LEVEL,id,APCY)  * Authorize each of the *
          ∪(id,EXPM)  IF  Auth_O(LEV,K_CUR_LEVEL,id,EXPM)  * eight modes of access *
          ∪(id,RDHS)  IF  Auth_O(K_CUR_LEVEL,LEV,id,RDHS)
          ∪(id,RDPM)  IF  Auth_O(K_CUR_LEVEL,LEV,id,RDPM)
          ∪(id,RDSZ)  IF  Auth_O(K_CUR_LEVEL,LEV,id,RDSZ)
          ∪(id,RETR)  IF  Auth_O(K_CUR_LEVEL,LEV,id,RETR)
          ∪(id,RSRV)  IF  Auth_O(LEV,K_CUR_LEVEL,id,RSRV) ∧ (LEV = K_CUR_LEVEL)
          ∪(id,STOR)  IF  Auth_O(LEV,K_CUR_LEVEL,id,STOR)

(ii) V-function  Auth_O(lv₁,lv₂,id,acc) : boolean

          * Check non-discretionary and discretionary access authorization. *

     range

          TRUE,FALSE

     parameter types

          level lv₁, level lv₂, identifier  id(OWN,NAM,TYP,LEV), access acc

     derivation

          (lv₁ ≻ lv₂) ∧ ((OWN = K_CUR_ID) ∨ ((K_CUR_ID,acc) ∈ D_M(id)))
          * non-discretionary AND ownership  OR  discretionary authorization. *
```

```
(iii)  V-function  Opened_O(id) : boolean

            * Return TRUE if any access mode is authorized. *

       range

            TRUE,FALSE

       parameter types

            identifier  id  (OWN,NAM,TYP,LEV)

       derivation

             Auth_O(LEV,K_CUR_LEVEL,id,APCY)
            vAuth_O(LEV,K_CUR_LEVEL,id,EXPM)
            vAuth_O(K_CUR_LEVEL,LEV,id,RDHS)
            vAuth_O(K_CUR_LEVEL,LEV,id,RDPM)
            vAuth_O(K_CUR_LEVEL,LEV,id,RDSZ)
            vAuth_O(K_CUR_LEVEL,LEV,id,RETR)
            vAuth_O(LEV,K_CUR_LEVEL,id,RSRV)  ∧  (LEV = K_CUR_LEVEL)
            vAuth_O(LEV,K_CUR_LEVEL,id,STOR)
```

O_APPEND


Subject 1
(at level W)
START
IL ?
N
Y
W_CODE ← IL
END
W ≯ LEV ?
Y
N
END
DO ?
N
Y
W_CODE ← DO
END
NE ?
N
Y
W_CODE ← NE
END
ND ?
N
Y
W_CODE ← DN
END
W_CODE ← ND
END
Subject 2
(at level W⌈id[LEV])
START
IL ?
N
Y
END
DO ?
N
Y
END
NE ?
N
Y
END
ND ?
N
Y
END
LEV ≯ W ?
N
Y
K_OPEN (id, access) = TRUE
END
Add user to D_O(id)
Update K_OPEN
END

PRIMITIVE: O_APPEND CASE: 1 SUBJECT: 1

CONDITIONS: IL
User's current level and object level are incomparable.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: IL	Unclass		
VARIABLES: K_CUR_LEVEL	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: O_APPEND CASE: 2 SUBJECT: 1

CONDITIONS: $(\sim IL) \wedge (W \not\propto LEV) \wedge DO$
Object is open already.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: DO	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN[LEV]	W LEV		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: O_APPEND CASE: 3 SUBJECT: 1

CONDITIONS: $(\sim IL) \wedge (W \succ LEV) \wedge (\sim DO) \wedge NE$
Non-existent object

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: NE	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN[LEV] D_E(id)	 W LEV LEV		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: O_APPEND CASE: 4 SUBJECT: 1

CONDITIONS: $(\sim IL) \wedge (W \succ LEV) \wedge (\sim DO) \wedge (\sim NE) \wedge ND$
User has no discretionary authorization.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: ND	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN[LEV] D_E(id) K_CUR_ID D_M(id)	 W LEV LEV W LEV		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: O_APPEND CASE: 5 SUBJECT: 1

CONDITIONS: $(\sim IL) \wedge (W \succ LEV) \wedge (\sim DO) \wedge (\sim NE) \wedge (\sim ND)$
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
id		W_CODE	W
CONSTANTS: APCY, EXPM, RDHS, RDPM, RDSZ, RETR, RSRV, STOR, DN	Unclass		
VARIABLES:			
K_CUR_LEVEL	W		
K_OPEN[LEV]	LEV		
D_E(id)	LEV		
K_CUR_ID	W		
D_M(id)	LEV		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: O_APPEND CASE: 1 SUBJECT: 2

CONDITIONS: $(\sim IL) \wedge (\sim DO) \wedge (\sim NE) \wedge (\sim ND) \wedge (LEV \not\succ W)$
No exceptions for a strictly dominated object.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
id		K_OPEN	W
CONSTANTS: APCY, EXPM, RDHS, RDPM, RDSZ, RETR, RSRV, STOR, DN	Unclass		
VARIABLES:			
K_CUR_LEVEL	W		
K_OPEN[LEV]	LEV		
D_E(id)	LEV		
K_CUR_ID	W		
D_M(id)	LEV		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: O_APPEND CASE 2 SUBJECT: 2

CONDITIONS: (~IL) ∧ (~DO) ∧ (~NE) ∧ (~ND) ∧ (LEV≻W)
No exceptions. Level of object being opened
dominates user's current level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:			
id	W	D_O(id)	LEV
CONSTANTS: APCY, EXPM, RDHS, RDPM, RDSZ, RETR, RSRV, STOR, DN	Unclass	K_OPEN[LEV]	LEV
VARIABLES:			
K_CUR_LEVEL	W		
K_OPEN[LEV]	LEV		
D_E(id)	LEV		
K_CUR_ID	W		
D_M(id)	LEV		
HIGHEST LEVEL OBSERVED:	LEV	LOWEST LEVEL MODIFIED:	LEV

A.1.3.5 APPEND

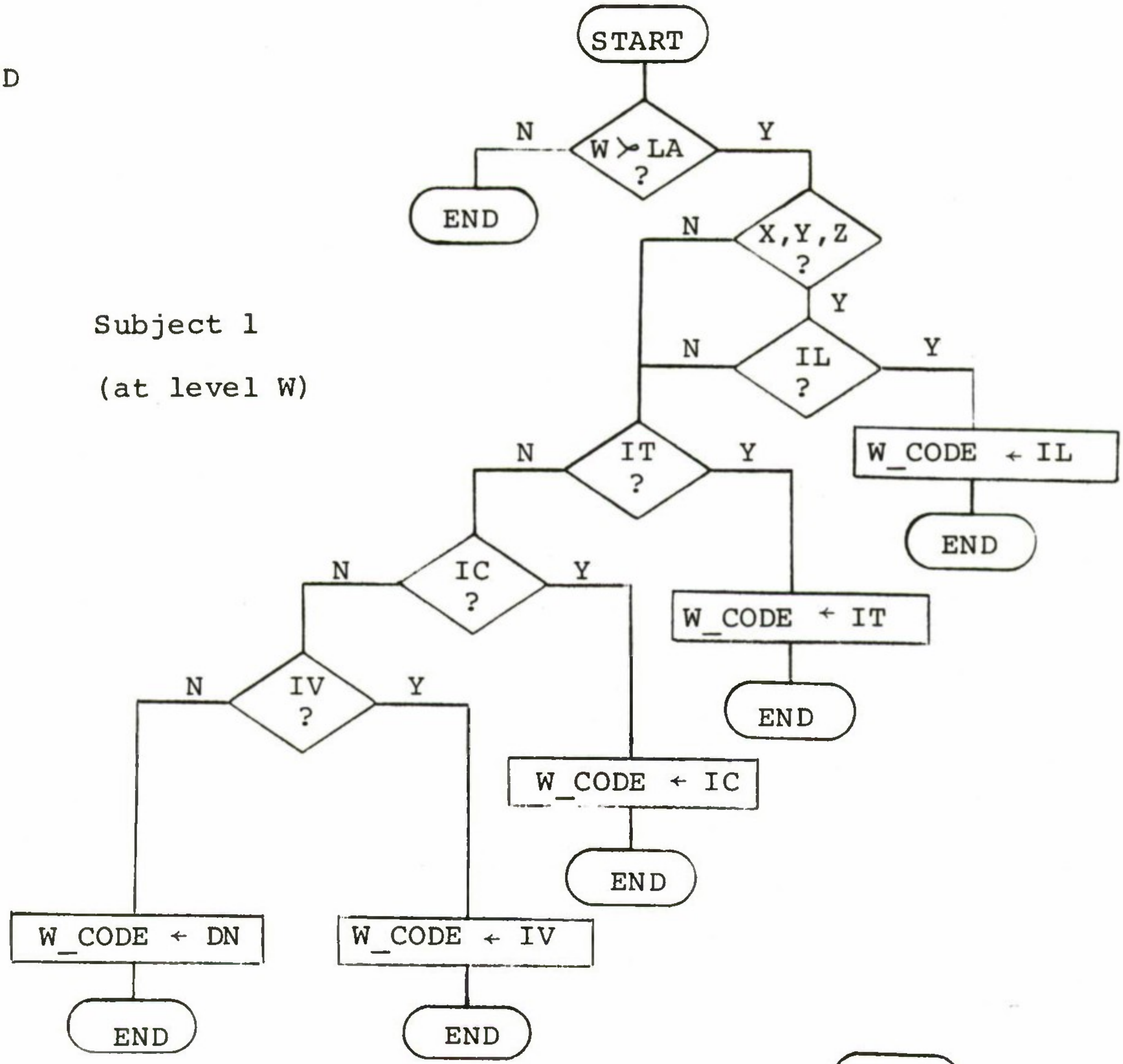
```
(A)  O-function  APPEND(xt)

          * The contents of "xt" are appended to the accumulator contents, where xt is: *
          *      (i)  a kernel temporary:  X; Y; or Z;                                  *
          * or  (ii)  a tuple of values from the working area.                          *
          * The APPEND fails if the primary key uniqueness property of the relation in  *
          * the accumulator would be destroyed.  This function differs from WW3 mainly  *
          * in that it is hidden, and only the accumulator is modified.                 *

     parameter types

          temp_tup xt, contents (O,N,T,L,C)            * Data in contents regs *

     abbreviation

          x = xt ∊ {X,Y,Z}           * x assumes TRUE/FALSE values *
     exception
        *IL[8]: K_LACC ⊁ K_Lxt   IF   x                 * Accumulator must dominate *
        *IT:  K_IACC[T] = 'S'                           * Don't do this for strings *
        *IC:  K_FACC[DTYPE] ≠ K_Fxt[DTYPE]  IF  x  ELSE  ~(xt CONFORMS TO K_FACC)
              * Tuple(s) to append must coincide domain by domain.                *
        *IV:  ~Unique_keys(K_FACC,⊢K_VACC⊣,K_Vxt)  IF x
                              ELSE  ~Unique_keys(K_FACC,⊢K_VACC⊣,xt)
              * Check that the primary key uniqueness property is maintained.     *


     effects

          [1]  K_VACC ← ⊢ ⊣ ∪ K_Vxt IF  x  ELSE ⊢ ⊣ ∪ xt
               * Append the temporary register or a tuple of parameters. *
         *[2]  W_CODE ← DN   IF  K_CUR_LEVEL ⊁ K_LACC              * Return code if possible *
```

(B) Access table

Variables Observed	Variables Modified	Variables Observed and Modified
K_IACC,K_Ixt K_FACC,K_Fxt K_Vxt,K_LACC,K_Lx	W_CODE	K_VACC

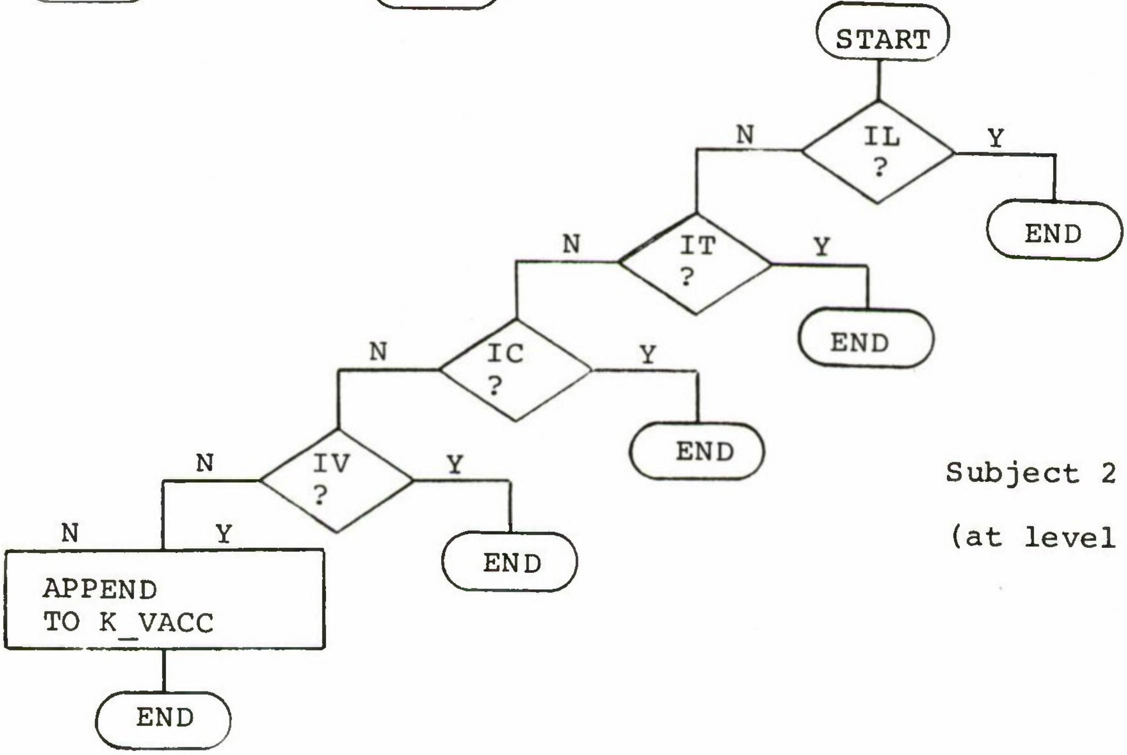
[8] The level of data in these boolean expressions is the level of the accumulator contents (not a parameter). This is required for setting W_CODE (for '*').

APPEND



Subject 1

(at level W)



Subject 2

(at level LA)

PRIMITIVE: APPEND　　　CASE: 1　　　SUBJECT: 1

CONDITIONS: (xt) ϵ {X,Y,Z}) $\wedge$ (W $\succ$ LA) $\wedge$ IL
xt is a kernel temporary. However, its level
is not dominated by the accumulator level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
xt(name of temporary)		W_CODE	W
CONSTANTS:	Unclass		
IL			
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_Lxt	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: W $\succ$ LA and Minimum K-level Invariant [c.f. § 4.1].

PRIMITIVE: APPEND　　　CASE: 2A　　　SUBJECT: 1

CONDITIONS: (xt ϵ {X,Y,Z}) $\wedge$ (W = LA) $\wedge$ (~IL) $\wedge$ IT
APPEND cannot be used with strings.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
xt(temporary)		W_CODE	W
CONSTANTS:	Unclass		
'S',IT			
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_Lxt	W		
K_IACC	LA = W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APPEND CASE: 2B SUBJECT: 1

CONDITIONS: (xt ∉ {X,Y,Z}) ∧ (W = LA) ∧ (~IL) ∧ IT
xt is a value in the user's working area.
Attempted to append this to a string in the accumulator.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt(value)	W	W_CODE	W
CONSTANTS: 'S',IT	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_IACC	 W W LA=W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	

PRIMITIVE: APPEND CASE: 3A SUBJECT: 1

CONDITIONS: (xt ∊ X,Y,Z) ∧ (W = LA) ∧ (~IL) ∧ (~IT) ∧ IC
xt is a kernel temporary. The domains of
xt do not conform to those in the accumulator.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt(temporary)	W	W_CODE	W
CONSTANTS: 'DTYPE','S',IC	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_Lxt K_IACC K_Fxt K_FACC	 W W W LA=W LX X		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LX = W

PROOF: (~IL) ⇒ LA $\succ$ LX
W = LA ⇒ W $\succ$ LX

PRIMITIVE: APPEND CASE: 3B SUBJECT: 1

CONDITIONS: $(xt \notin \{X,Y,Z\}) \wedge (W = LA) \wedge (\sim IL) \wedge (\sim IT) \wedge IC$
xt is a user working area value which does
not conform to the accumulator contents.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt(value	W	W_CODE	W
CONSTANTS: 'DTYPE','S',IC	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_IACC K_FACC K_Fxt	 W W LA=W W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APPEND CASE: 4A SUBJECT: 1

CONDITIONS: $(xt \in \{X,Y,Z\}) \wedge (W = LA) \wedge (\sim IL) \wedge (\sim IT) \wedge (\sim IC) \wedge IV$
xt is a temporary, and appending its tuples
to the accumulator would result in duplicate keys.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt(temporary)	W	W_CODE	W
CONSTANTS: 'DNAME','DTYPE' 'WIDTH','ROLE','S',IV	Unclass		

VARIABLES:
K_CUR_LEVEL
K_LACC
K_Lxt
K_IACC
K_Fxt
K_FACC
K_Vxt
K_VACC
	
	W
	W
	W
	LA=W
	LA=W
	LA=W
	LX=W
	LA=W		

HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

See Case 3A for Subject 1

PRIMITIVE: APPEND CASE: 4B SUBJECT: 1

CONDITIONS: $(xt \notin \{X,Y,Z\}) \wedge (W = LA) \wedge (\sim IL) \wedge (\sim IC) \wedge IV$
xt is a user working area value, but appending it to the accumulator would produce dulicate keys.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	xt(value	W	W_CODE	W
CONSTANTS:	'DNAME','DTYPE' 'WIDTH','ROLE','S',IV	Unclass		
VARIABLES:	K_CUR_LEVEL	W		
	K_LACC	W		
	K_IACC	LA=W		
	K_Fxt	W		
	K_FACC	LA=W		
	K_Vxt	W		
	K_VACC	LA=W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APPEND CASE: 5A SUBJECT: 1

CONDITIONS: $(xt \in \{X,Y,Z\}) \wedge (W = LA) \wedge (\sim IL) \wedge (\sim IT) \wedge (\sim IC) \wedge (\sim IV)$
xt is a temporary, and there are no exceptions.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	xt	W	W_CODE	W
CONSTANTS:	'DNAME','DTYPE' 'WIDTH','ROLE','S',DN	Unclass		
VARIABLES:	K_CUR_LEVEL	W		
	K_LACC	W		
	K_IACC	LA=W		
	K_Lxt	W		
	K_Fxt	LA=W		
	K_FACC	LA=W		
	K_Vxt	LX=W		
	K_VACC	LA=W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APPEND   CASE: 5B   SUBJECT: 1

CONDITIONS: $(xt \notin X,Y,Z) \land (W = LA) \land (\sim IL) \land (\sim IT) \land (\sim IC) \land (\sim IV)$
xt is a user working area value, and
there are no exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt(value)	W	W_CODE	W
CONSTANTS: 'DNAME',DTYPE', 'WIDTH','ROLE','S',DN	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_IACC K_Fxt K_FACC K_Vxt K_VACC	W W LA=W W LA=W W LA=W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: APPEND   CASE: 6   SUBJECT: 1

CONDITIONS: $W \not> LA$

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt	W		W[9]
CONSTANTS:	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC	W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

[9] The "null" return code is at level W.

PRIMITIVE: APPEND CASE: 1A SUBJECT: 2

CONDITIONS: $(xt \in \{X,Y,Z\}) \wedge (\sim IL) \wedge (\sim IT) \wedge (\sim IC) \wedge (\sim IV)$
xt is a temporary.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt(temporary)	W	K_VACC	LA
CONSTANTS: 'DNAME','DTYPE' 'WIDTH','ROLE','S',DN	Unclass		
VARIABLES: K_CUR_LEVEL	W		
K_LACC	W		
K_Lxt	W		
K_IACC	LA		
K_Fxt	LX		
K_FACC	LA		
K_Vxt	LA		
K_VACC	LA		
HIGHEST LEVEL OBSERVED:	LA	LOWEST LEVEL MODIFIED:	LA

LEMMA: LA $\succ$ W
PROOF: Minimum K-level Invariant
[c.f. § 4.1]

LEMMA: LA $\succ$ LX
PROOF: ~IL

PRIMITIVE: APPEND CASE: 1B SUBJECT: 2

CONDITIONS: $(xt \notin X,Y,Z) \wedge (\sim IL) \wedge (\sim IT) \wedge (\sim IC) \wedge (\sim IV)$
xt is a user working area value.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: xt(value)	W	K_VACC	LA
CONSTANTS: 'DNAME','DTYPE', 'WIDTH','ROLE','S',DN	Unclass		
VARIABLES: K_CUR_LEVEL	W		
K_LACC	W		
K_IACC	LA		
W_Fxt	W		
K_FACC	LA		
W_Vxt	W		
K_VACC	LA		
HIGHEST LEVEL OBSERVED:	LA	LOWEST LEVEL MODIFIED:	LA

LEMMA: LA $\succ$ W
PROOF: Minimum K-level Invariant

A.1.3.6 CONCAT

```
(A)  O-function  CONCAT(x)

          * Concatenate a string in a temporary variable to the string in the *
          * accumulator.  Only fields with unique names will be concatenated. *

     parameter types

          temp x, format(DNAME,DTYPE,WIDTH,ROLE), contents(O,N,T,L,C)

     abbreviation

          common = ⊢K_FACC[DNAME]⊣ ∩ K_Fx[DNAME]        * Field names in common *

     exceptions

         *IL:  K_LACC ⊁ K_Lx                             * Append must be "up"
         *IV:  (K_IACC[T;C] ≠ ('S','V')) ∨ (K_Ix[T;C] ≠ ('S','V'))
               * Check that these are really strings. *
         *ND:  K_Fx[DNAME] ⊆ ⊢K_FACC[DNAME]⊣             * There are no new fields *

     effects

          [1]  K_FACC ← ⊢ ⊣ ∪ (K_Fx - K_Fx{DNAME ε common})
               * Append format tuples for fields with unique names. *
          [2]  K_VACC ← ⊢ ⊣ ∪ (K_Vx - K_Vx[common])
               * Append value tuples for fields with unique names.  *
         *[3]  W_CODE ← DN  IF  K_CUR_LEVEL ≻ K_LACC
```

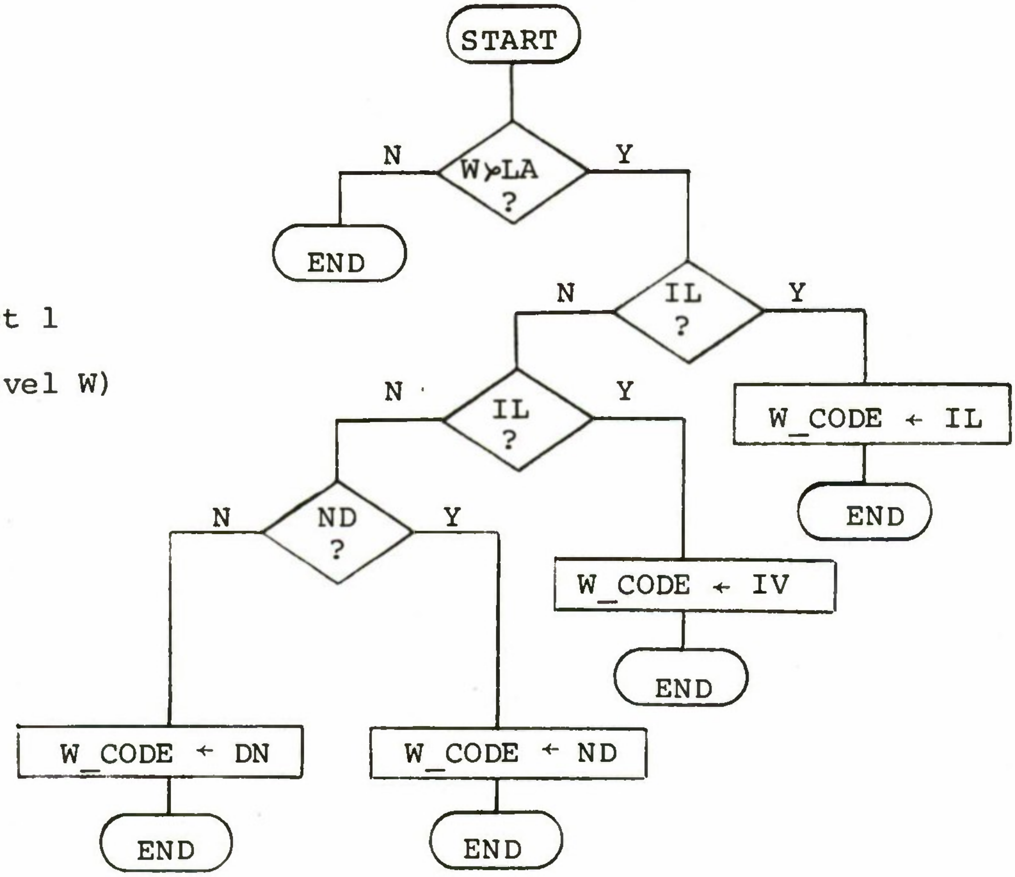
(B) Access table

Variables Observed	Variables Modified	Variables Observed and Modified
K_IACC,K_Ix	W_CODE	
K_Fx,K_Vx		K_VACC
K_LACC,K_Lx		K_FACC

CONCAT

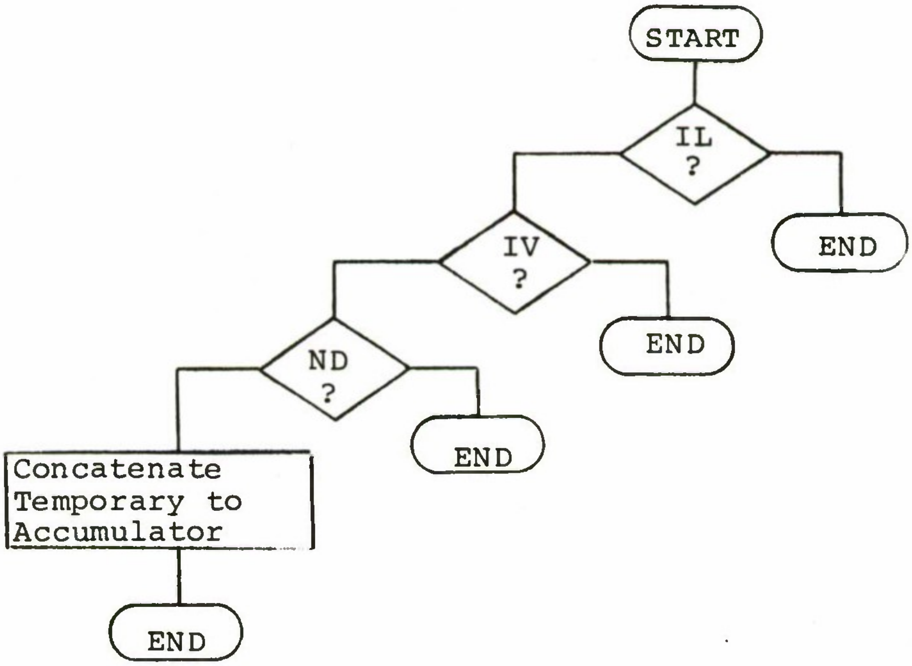
Subject 1

(at level W)



Subject 2

(at level LA)



PRIMITIVE: CONCAT CASE: 1 SUBJECT: 1

CONDITIONS: $(W \nsucc LA) \wedge IL$
Level of temporary is not dominated
by the level of the accumulator.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
x(temporary)		W_CODE	W
CONSTANTS:	Unclass		
IL			
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_Lx	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA=W
PROOF: $W \succ LA$
$LA \nsucc W$ by Minimum K-level Invariant [c.f. § 4.1]

PRIMITIVE: CONCAT CASE: 2 SUBJECT: 1

CONDITIONS: $(W = LA) \wedge (\sim IL) \wedge IV$
Accumulator and temporary do not
bot contain strings.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
x		W_CODE	W
CONSTANTS:	Unclass		
IV			
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_Lx	W		
K_IACC	LA=W		
K_Ix	LX		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LX = W
PROOF: $LA \succ LX$ By $\sim IL \Rightarrow W \nsucc LX$
$LX \nsucc W$ By Minimum K-level Invariant

PRIMITIVE: CONCAT **CASE:** 3 **SUBJECT:** 1

CONDITIONS: (W = LA) ∧ (~IL) ∧ (~IV) ∧ ND
There are no new fields in the string to be concatenated.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	x	W	W_CODE	W
CONSTANTS:	'DNAME',ND	Unclass		
VARIABLES:	K_CUR_LEVEL K_LACC K_Lx K_IACC K_Ix K_FACC K_Fx	W W W LA=W LX LA=W LX		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

LEMMA: LX = W
PROOF: W = LA and LA $\succeq$ LX By ~IL ⇒ W $\succeq$ LX
LX $\succeq$ W By Minimum K-level Invariant [c.f. § 4.1]

PRIMITIVE: CONCAT **CASE:** 4 **SUBJECT:** 1

CONDITIONS: (W = LA) ∧ (~IL) ∧ (~IV) ∧ (~ND)
No exceptions.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	x	W	W_CODE	W
CONSTANTS:	'DNAME',DN	Unclass		
VARIABLES:	K_CUR_LEVEL K_LACC K_Lx K_IACC K_Ix K_FACC K_Fx	W W W LA=W LX LA=W LX		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

LX = W By LEMMA in Case 3.

PRIMITIVE: CONCAT CASE: 5 SUBJECT: 1

CONDITIONS: $(W \not\succ LA)$
User's current level does not dominate accumulator.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: x	W		W
CONSTANTS:	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC	 W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: CONCAT CASE: 1 SUBJECT: 2

CONDITIONS: $(\sim IL) \wedge (\sim IV) \wedge (\sim ND)$
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: x	W	K_FACC K_VACC	LA LA
CONSTANTS: 'DNAME',DN	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_Lx K_IACC K_Ix K_FACC K_Fx	W W W LA LX LA LX		
HIGHEST LEVEL OBSERVED:	LA	LOWEST LEVEL MODIFIED:	LA

LEMMA: $LX \succ W$
PROOF: Minimum K-level Invariant

LEMMA: $LA \succ LX$
PROOF: ~IL

A.1.3.7 DKD

```
(A)  O-function  DKD(lv)

          * Copy a data base directory to the kernel accumulator. *

     parameter types

          level lv

     exception

          IL:  K_CUR_LEVEL ≯ lv                    * Current level must dominate *
          NF:  D_D(lv) = Ø                         * The directory is not found  *

     effect
          [1]  K_LACC ← K_CUR_LEVEL                * This is "minimum" level     *
          [2]  K_FACC ← ('OWNER','I',USE_WIDTH,1) ∪ ('NAME','C',MAX_NAME,2)    * continued *
                      ∪ ('TYPE','C',1,3) ∪ ('LEVEL','I',LEV_WIDTH,4)
          [3]  K_IACC ← (K_CUR_ID,'D','R',lv,'V')      * Identify the data *
          [4]  K_VACC ← D_D(lv)                    * Now copy the directory data *
          [5]  W_CODE ← DN
```

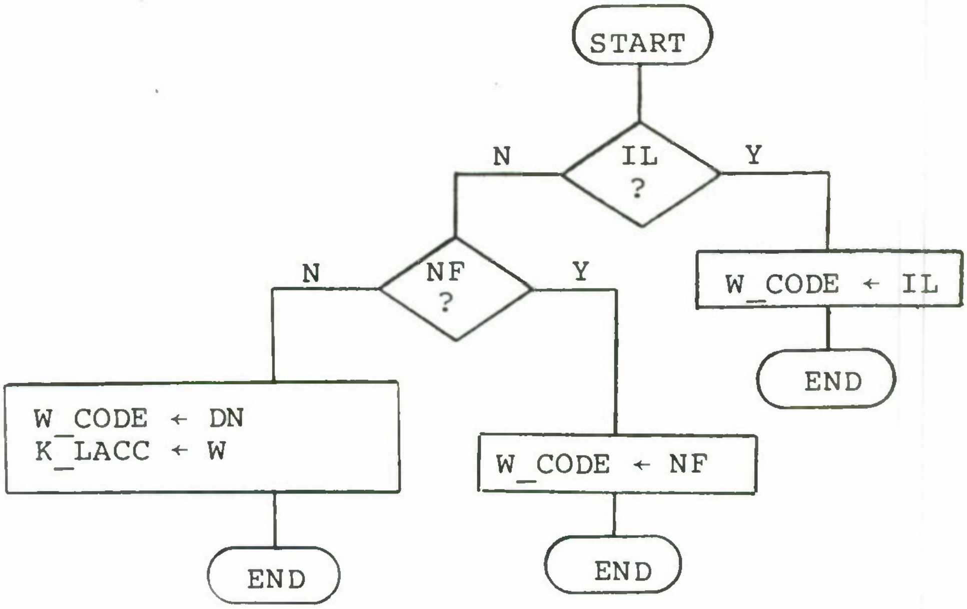
(B) Access table

Variables Observed	Variables Modified	Variables Observed and Modified
K_CUR_LEVEL	K_FACC	
D_D(level)	K_IACC	
K_CUR_ID	K_VACC	
	K_LACC	
	W_CODE	

DKD

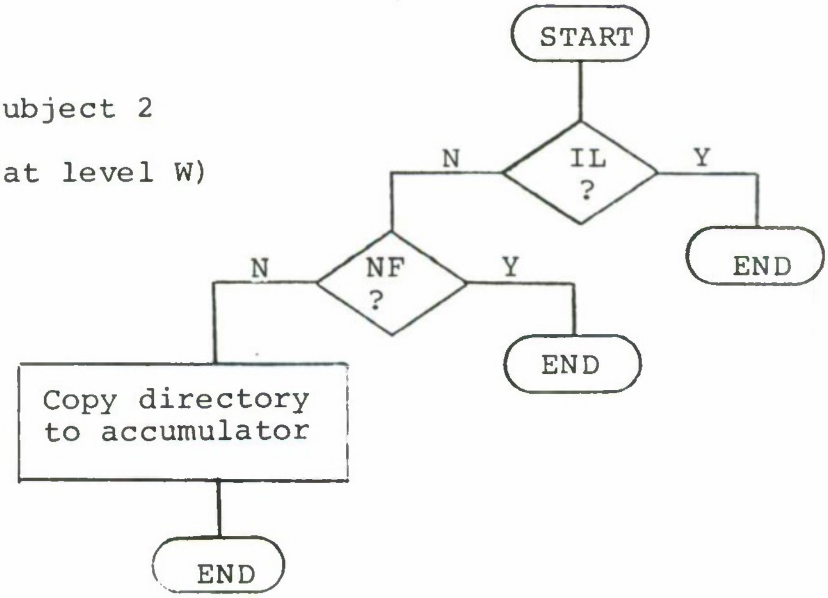
Subject 1

(at level W)



Subject 2

(at level W)



PRIMITIVE: DKD CASE: 1 SUBJECT: 1

CONDITIONS: IL
User's current level does
dominate the directory level.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	lv	W	W_CODE	W
CONSTANTS:	IL	Unclass		
VARIABLES:	K_CUR_LEVEL	W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: DKD CASE: 2 SUBJECT: 1

CONDITIONS: (~IL) ∧ NF
No directory exists at level lv.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	lv	W	W_CODE	W
CONSTANTS:	Ø, NF	Unclass		
VARIABLES:	D_D(lv)	lv		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

LEMMA: $W \not\succ lv$
PROOF: ~IL

PRIMITIVE: DKD CASE: 3 SUBJECT: 1

CONDITIONS: (~IL) ∧ (~NF)
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
lv		W_CODE K_LACC	W W
CONSTANTS: ∅,1,2,3,4,'OWNER', 'NAME','TYPE','LEVEL',USE_WIDTH, MAX_NAME,LEV_WIDTH,'R','V',DN	Unclass		
VARIABLES:			
D_D(lv) K_CUR_ID	lv W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: $W \succ lv$
PROOF: ~IL

PRIMITIVE: DKD CASE: 1 SUBJECT: 2

CONDITIONS: (~IL) ∧ (~NF)
No exceptions. Copy directory into accumulator.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
lv		K_FACC K_IACC K_VACC	LA LA LA
CONSTANTS: (Same as case 3)	Unclass		
VARIABLES:			
D_D(lv) K_CUR_ID	lv W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: $W \succ lv$
PROOF: ~IL

LEMMA: LA = W
PROOF: Effect[1] of specification

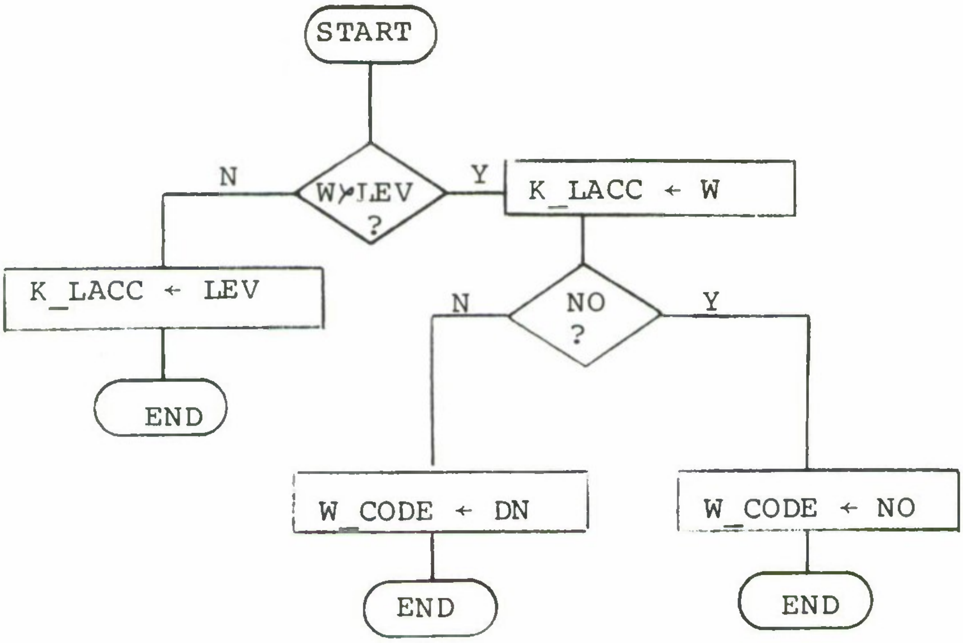
A.1.3.8 DKE

```
(A)  O-function  DKE(id)

          * Copy the specified exact size component to the kernel accumulator. *
          * The object must be open, in order to justify this data movement.   *

     parameter types

          identifier id (OWN,NAM,TYP,LEV)

     exception

        **NO:  K_OPEN(id,*) = FALSE                   * The object is not open *
     effects

          [1]  K_LACC ← LEV  IF  LEV ≻ K_CUR_LEVEL  ELSE  K_CUR_LEVEL
          [2]  K_FACC ← ('EXACT','I',SIZ_WIDTH,1)     * Set format for a single value    *
          [3]  K_IACC ← (id,'E')                      * Identify the accumulator contents *
          [4]  K_VACC ← D_E(id)                       * Copy the exact size value         *
         *[5]  W_CODE ← DN  IF  K_CUR_LEVEL ≻ LEV     * Return code if a                  *
```

(B) Access table

Variables Observed	Variables Modified	Variables Observed and Modified
K_OPEN	K_FACC	
K_CUR_LEVEL	K_IACC	
D_E(id)	K_VACC	
	W_CODE	
	K_LACC	

DKE

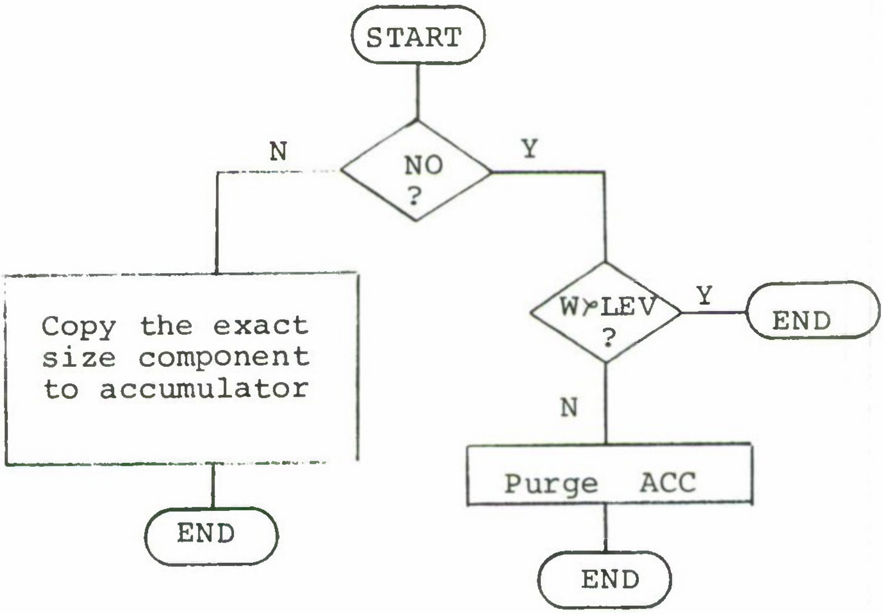


Subject 1

(at level W)

Subject 2

(at level W⌈LEV)



PRIMITIVE: DKE CASE: 1 SUBJECT: 1

CONDITIONS: $W \not\succ LEV$
Level of object strictly dominates

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	id	W	K_LACC	W
CONSTANTS:		Unclass		
VARIABLES:	K_CUR_LEVEL	W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: DKE CASE: 2 SUBJECT: 1

CONDITIONS: $(W \succ LEV) \wedge NO$
Object has not been opened.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	id	W	K_LACC W_CODE	W W
CONSTANTS:	NO,FALSE	Unclass		
VARIABLES:	K_CUR_LEVEL K_OPEN[LEV]	W W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: DKE CASE: 3 SUBJECT: 1

CONDITIONS: $(W \not\succ LEV) \wedge (\sim NO)$
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	K_LACC W_CODE	W W
CONSTANTS: DN,FALSE	Unclass		
VARIABLES: K_CUR_LEVEL K_OPEN[LEV]	W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: DKE CASE: 1 SUBJECT: 2

CONDITIONS: $NO \wedge (W \not\succ LEV)$
Dominated object is not open.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W		W
CONSTANTS: FALSE	Unclass		
VARIABLES: K_OPEN[LEV] K_CUR_LEVEL	W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: DKE CASE: 2 SUBJECT: 2

CONDITIONS: NO ∧ $(W \not\succ LEV)$
Strictly dominating object is not open.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	id	W	K_FACC K_IACC K_VACC	LEV LEV LEV
CONSTANTS:	Ø,FALSE	Unclass		
VARIABLES:	K_CUR_LEVEL K_OPEN[LEV]	W LEV		
HIGHEST LEVEL OBSERVED:		LEV	LOWEST LEVEL MODIFIED:	LEV

LEMMA: $LEV \succ W$
PROOF: Minimum K-level Invariant

PRIMITIVE: DKE CASE: 3 SUBJECT: 2

CONDITIONS: ~NO
Object is open, no exception.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	id	W	K_FACC K_IACC K_VACC	W⌈LEV W⌈LEV W⌈LEV
CONSTANTS:	FALSE,'EXACT','I', SIZ_WIDTH,'E'	Unclass		
VARIABLES:	K_CUR_LEVEL K_OPEN[LEV]	W W⌈LEV		
HIGHEST LEVEL OBSERVED:		W⌈LEV	LOWEST LEVEL MODIFIED:	W⌈LEV

A.1.3.9 KDM

```
(A)  O-function  KDM(id)

          * Copy an access permission matrix from the kernel accumulator *
          * to the data base.  This is required for EXTEND-PERMISSION    *

     parameter types

          identifier id (OWN,NAM,TYP,LEV), history (CREATION,USER,MODIFICATION)

     abbreviation       * This is a proper format for a permission matrix. *

          proper_format = ('USER','I',USE_WIDTH,1) ∪ ('VISIBLE','L',1,0)
     exceptions

          IL:  K_LACC ≠ LEV                              * Accumulator level is wrong   *
         *NO:  K_OPEN(id,EXPM) = FALSE                   * Object is not open           *
         *IC:  K_IACC ≠ (id,'M')                         * It's not the permission matrix *
         *IV:  K_FACC ≠ proper_format                    * It's not in proper format    *
         *SZ:  (⊲(K_VACC) + ⊲(D_F(id)) + ⊲(D_V(id))) > D_Z(id)  * Blows space limit  *
               * ⊲ is a system function to compute the size of something.            *

     effect

          [1]  D_M(id) ← K_VACC                          * Copy the new permission matrix *
          [2]  D_E(id) ← ⊲(K_VACC) + ⊲(D_F(id)) + ⊲(D_V(id))    * Compute new size      *
          [3]  D_H(id) ← (⊢⊣[CREATION],K_CUR_ID,K_CUR_TIME)
                      * Copy the creation date, and construct rest of history          *
         *[4]  W_CODE ← DN   IF  K_CUR_LEVEL = LEV
```

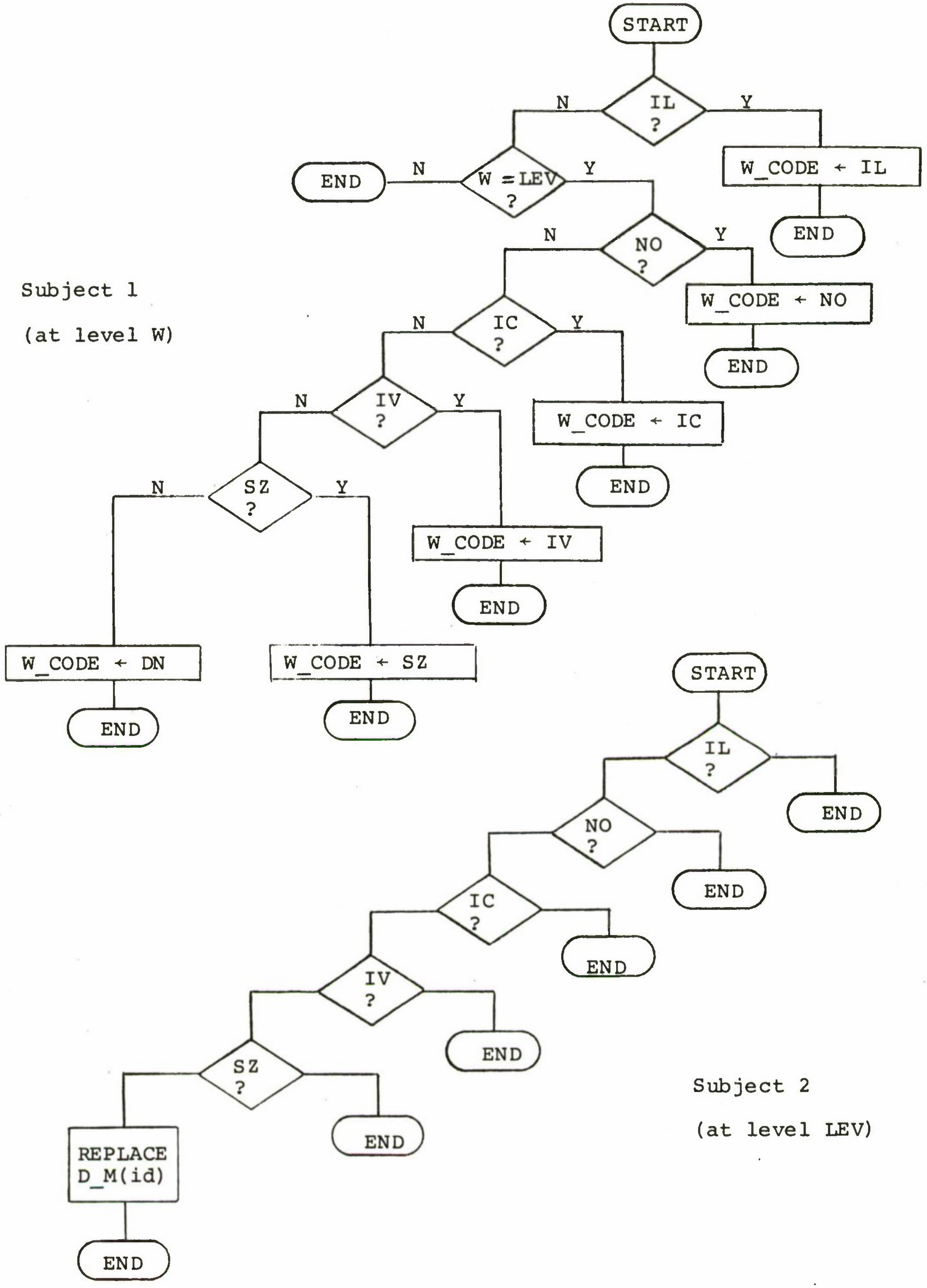
(B) Access table

	Variables Observed	Variables Modified	Variables Observed and Modified
	K_OPEN,K_CUR_ID	D_M(id)	D_H(id)
	K_FACC,K_IACC,K_VACC	D_E(id)	
	D_F(id),D_V(id),D_Z(id)	W_CODE	
	K_CUR_LEVEL K_CUR_TIME		
	K_LACC		

KDM

Subject 1

(at level W)



Subject 2

(at level LEV)

PRIMITIVE: KDM CASE: 1 SUBJECT: 1

CONDITIONS: IL
Accumulator is not at proper level (W).

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: IL	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC	 W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDM CASE: 2 SUBJECT: 1

CONDITIONS: (~IL) ∧ NO ∧ (W = LEV)
Dominated object is not open.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: NO	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_OPEN[LEV]	 W W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDM CASE: 3 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W = LEV) ∧ (~NO) ∧ IC
The accumulator does not contain
the object's permission matrix.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
id		W_CODE	W
CONSTANTS:	Unclass		
IC			
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_OPEN[LEV]	W		
K_IACC	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDM CASE: 4 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W = LEV) ∧ (~NO) ∧ (~IC) ∧ IV
Format of accumulator contents is inappropriate.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: 0,1,'I','L','M',IV, 'USER','VISIBLE',USE_WIDTH	Unclass		
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_OPEN[LEV]	W		
K_IACC	W		
K_FACC	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDM CASE 5 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W = LEV) ∧ (~NO) ∧ (~IC) ∧ (~IV) ∧ SZ
This permission matrix would cause the
maximum size of this object to be exceeded.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: o,l,'I','L','M',SZ 'USER','VISIBLE',USE_WIDTH	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_OPEN[LEV] K_IACC K_FACC D_F(id) D_V(id) D_Z(id)	 W W LA LA LA LEV=W LEV=W LEV=W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: (~IL) ∧ (W = LEV)

PRIMITIVE: KDM CASE 6 SUBJECT: 1

CONDITIONS: (~IL) ∧ (~NO) ∧ (~IC) ∧ (~IV) ∧ (~SZ) ∧ (W = LEV)
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:			
id	W	W_CODE	W
CONSTANTS: 0,1,'I','L','M',DN 'USER','VISIBLE',USE_WIDTH	Unclass		
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_OPEN[LEV]	W		
K_IACC	W		
K_FACC	W		
D_F(id)	LEV=W		
D_V(id)	LEV=W		
D_Z(id)	LEV=W		
K_CUR_TIME	W		
D_H(id)	LEV=W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDM CASE 1 SUBJECT: 2

CONDITIONS: (~IL) ∧ (~NO) ∧ (~IC) ∧ (~IV) ∧ (~SZ)
No exception.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:			
id	W	D_M(id)	LEV
		D_E(id)	LEV
		D_H(id)	LEV
CONSTANTS: 0,1,'I','L','M' 'USER','VISIBLE',USE_WIDTH	Unclass		
VARIABLES:			
K_CUR_LEVEL	W		
K_LACC	W		
K_OPEN[LEV]	LEV		
K_IACC	LEV		
K_FACC	LEV		
D_F(id)	LEV		
D_V(id)	LEV		
D_Z(id)	LEV		
K_CUR_TIME	W		
D_H(id)	LEV		
HIGHEST LEVEL OBSERVED:	LEV	LOWEST LEVEL MODIFIED:	LEV

LEMMA: LEV $\succ$ W

PROOF: ~NO ⇒ K_OPEN(id,EXPM) = TRUE
The derivation of the Access_set_O and Auth_O
V-functions in O_APPEND complete the proof.

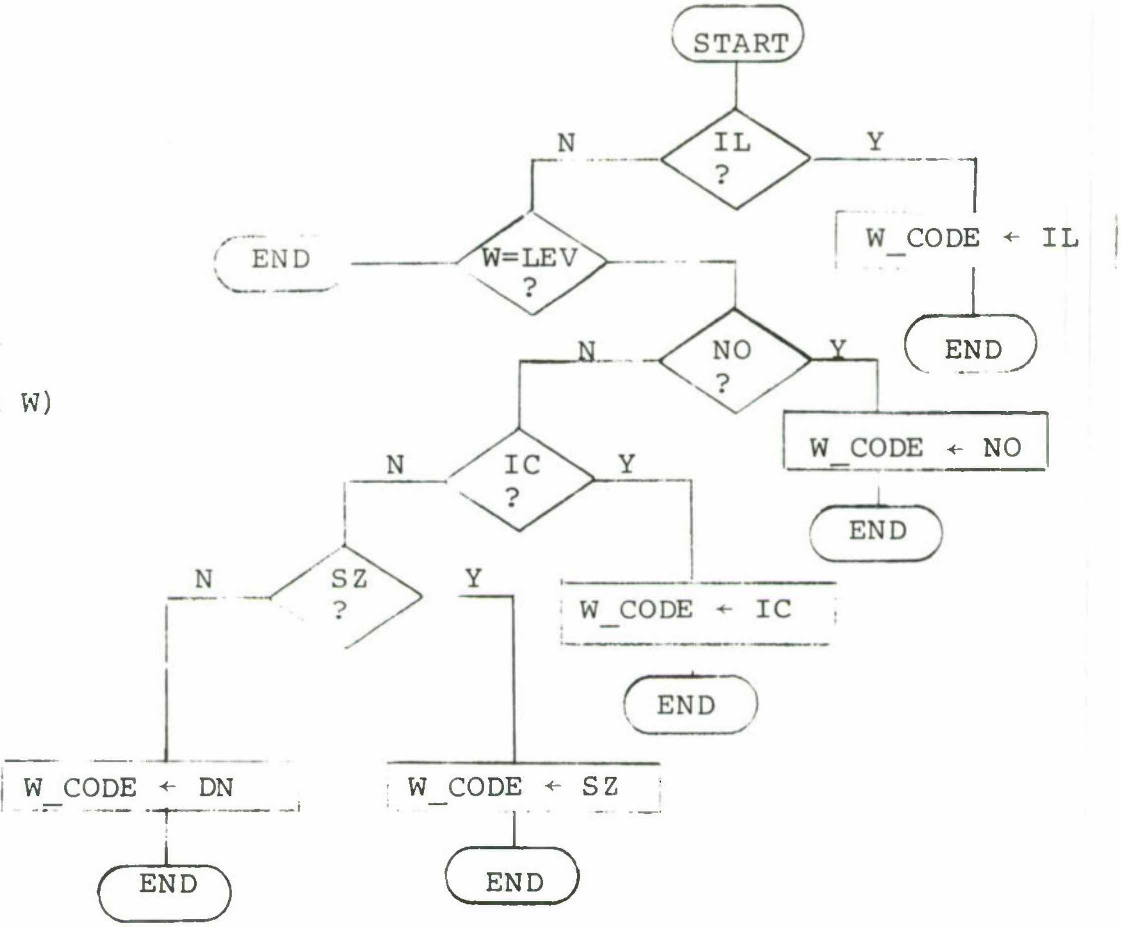
A.1.3.10 KDV

```
(A)  O-function  KDV(id)

          * Copy an object's format and values from *
          * the accumulator to the data base        *

     parameter types

          identifier id (OWN,NAM,TYP,LEV), history (CREATION,USER,MODIFIED)

     exceptions

          IL:  K_LACC ≠ LEV                               * Accumulator level is wrong       *
         *NO:  K_OPEN(id,STOR) = FALSE                    * The object is NOT open.          *
         *IC:  K_IACC ≠ (id,'V')                          * Incorrect accumulator contents.  *
         *SZ:  (⍴(D_M(id)) + ⍴(K_FACC) + ⍴(K_VACC)) > D_Z(id)    * Not enough space        *

     effects

          [1]  D_E(id) ← ⍴(D_M(id)) + ⍴(K_FACC) + ⍴(K_VACC)      * Reset current size *
          [2]  D_H(id) ← (⊢ ⊣[CREATION],K_CUR_ID,K_CUR_TIME)     * Update the history *
          [3]  D_F(id) ← K_FACC                           * Copy the format          *
          [4]  D_V(id) ← K_VACC                           * Copy the table of values *
         *[5]  W_CODE ← DN  IF  K_CUR_LEVEL = LEV
```

(B) Access table

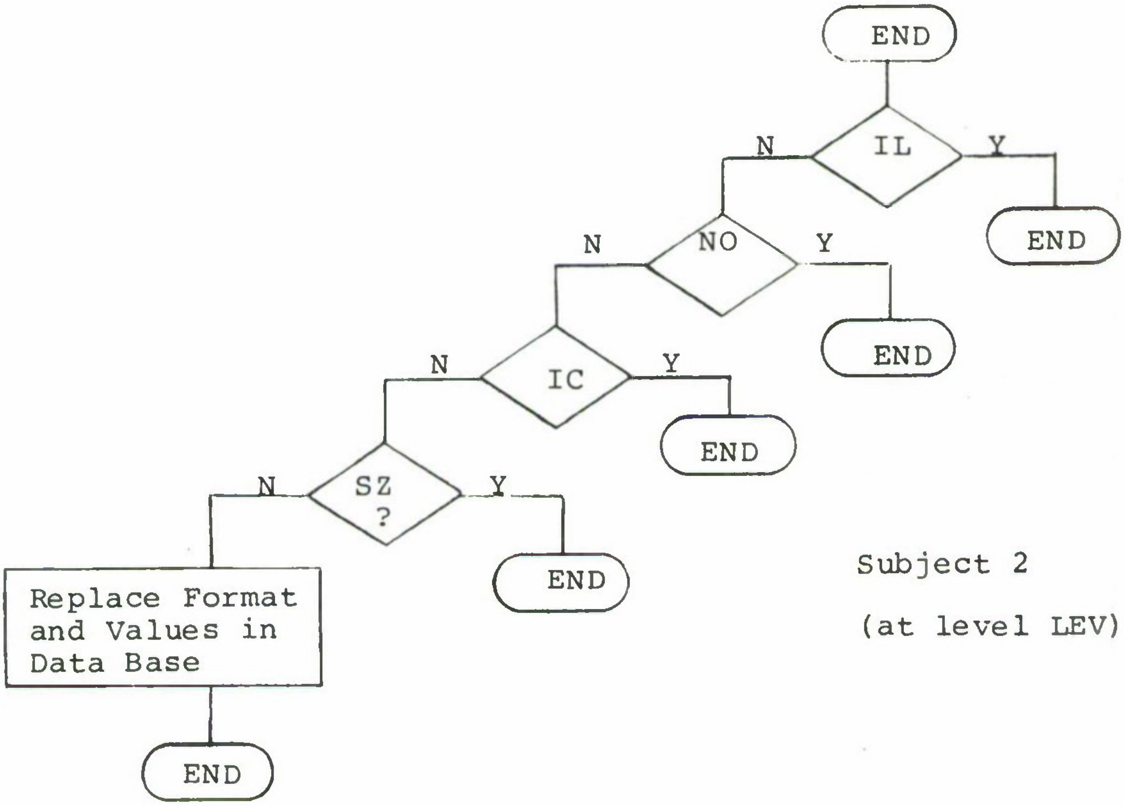
Variables Observed	Variables Modified	Variables Observed and Modified
K_OPEN	D_E(id)	D_H(id)
K_IACC,K_LACC		
D_M(id),D_Z(id)	D_F(id)	
K_FACC,K_VACC	D_V(id)	
K_CUR_ID	W_CODE	
K_CUR_TIME		
K_CUR_LEVEL		

KDV



Subject 1

(at level W)



Subject 2

(at level LEV)

PRIMITIVE: KDV CASE: 1 SUBJECT: 1

CONDITIONS: IL
Accumulator level is not equal to object level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W	W_CODE	W
CONSTANTS: IL	Unclass		
VARIABLES: K_LACC	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDV CASE: 2 SUBJECT: 1

CONDITIONS: $(\sim IL) \wedge (W \neq LEV)$
Object being replaced is at
a strictly dominating level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id	W		W
CONSTANTS:	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC	 W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDV CASE: 3 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W = LEV) ∧ NO
Object is not open with STOR access.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
id		W_CODE	W
CONSTANTS:	Unclass		
STOR,NO			
VARIABLES:			
K_LACC	W		
K_CUR_LEVEL	W		
K_OPEN[LEV]	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDV CASE: 4 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W = LEV) ∧ (~NO) ∧ IC
Accumulator does not contain
the appropriate value set.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
id		W_CODE	W
CONSTANTS:	Unclass		
'V',STOR,IC			
VARIABLES:			
K_LACC	W		
K_CUR_LEVEL	W		
K_OPEN[LEV]	LEV=W		
K_IACC	LA		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL ⇒LA = LEV
But LEV = W

PRIMITIVE: KDV CASE 5 SUBJECT: 1

CONDITIONS: (~IL) ^ (W = LEV) ^ (~NO) ^ (~IC) ^ SZ
User has insufficient space to store
the value set in the data base.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:			
id	W	W_CODE	W
CONSTANTS:	Unclass		
'V',STOR,SZ			
VARIABLES:			
K_LACC	W		
K_OPEN[LEV]	W		
K_IACC	LA		
D_M(id)	LEV=W		
K_VACC	LA		
D_Z(id)	LEV=W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL => LA = LEV
But LEV = W

PRIMITIVE: KDV CASE 6 SUBJECT: 1

CONDITIONS: (~IL) ∧ (W = LEV) ∧ (~NO) ∧ (~IC) ∧ (~SZ)
No exceptions.

OBSERVED		LEVEL	MODIFIED	LEVEL
PARAMETERS:				
	id	W	W_CODE	W
CONSTANTS:				
	'V',STOR,DN	Unclass		
VARIABLES:				
	K_LACC	W		
	K_OPEN[LEV]	W		
	K_IACC	LA		
	D_M(id)	LEV=W		
	K_FACC	LA		
	K_VACC	LA		
	D_Z(id)	LEV=W		
	D_H(id)	LEV=W		
	K_CUR_ID	W		
	K_CUR_LEVEL	W		
	K_CUR_TIME	W		
HIGHEST LEVEL OBSERVED:		W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL => LA = LEV
But LEV = W

PRIMITIVE: KDV CASE 1 SUBJECT: 2

CONDITIONS: (~IL) ∧ (~NO) ∧ (~IC) ∧ (~SZ)
No exceptions. Replace value set in data base.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:			

id	W	D_E(id)
		D_V(id)
		D_F(id)
			LEV
			LEV
			LEV

CONSTANTS:	Unclass		
'V',STOR			
VARIABLES:			
K_LACC	W		
K_OPEN[LEV]	LEV		
K_IACC	LA		
K_FACC	LA		
K_VACC	LA		
D_M(id)	LEV		
D_Z(id)	LEV		
D_H(id)	LEV		
K_CUR_ID	W		
K_CUR_TIME	W		
K_CUR_LEVEL	W		
HIGHEST LEVEL OBSERVED:	LEV	LOWEST LEVEL MODIFIED:	LEV

LEMMA: LEV $\succeq$ W
PROOF: K_OPEN(id,EXPM) = TRUE by ~NO
The derivations of the
Access_set_O and Auth_O
V-functions in O_APPEND
complete the proof.

LEMMA: LA = LEV
PROOF: ~IL

A.1.3.11 KDZ

```
(A)  O-function  KDZ(n,t)

          * Resize an object (maximum space) in the data base by copying  *
          * its maximum size value from the accumulator to the data base. *
          * The session quota is updated appropriately.  The object       *
          * needn't be open, since this can be done ONLY by an object's   *
          * owner, at the current level.                                  *

     parameter types

          name n, type t, level lv

     abbreviation

          id     = K_CUR_ID,n,t,K_CUR_LEVEL           * Identify the data base object     *
          change = K_VACC - ├D_Z(id)┤                 * This is the amount of size change *

     exception

          IL:  K_CUR_LEVEL ≠ K_LACC                   * Accumulator level is wrong     *
          NE:  D_Z(id) = Ø                            * Component object doesn't exist *
          IC:  K_IACC ≠ (id,'Z')                      * It is not a maximum size       *

          SZ:  (K_VACC < D_E(id)) ∨ (change >  K_CUR_QTA     * Size error if              *
               * new size is less than current OR change is more than session quota       *

     effect

          [1]  D_Z(id) ← K_VACC                       * Set the new maximum size *
          [2]  K_CUR_QTA ← ├ ┤ - change               * Update session quota     *
          [3]  W_CODE ← DN
```

(B) Access table

Variables Observed	Variables Modified	Variables Observed and Modified
K_CUR_ID K_VACC,K_IACC K_CUR_LEVEL D_E(id) K_LACC	W_CODE	D_Z(id) K_CUR_QTA

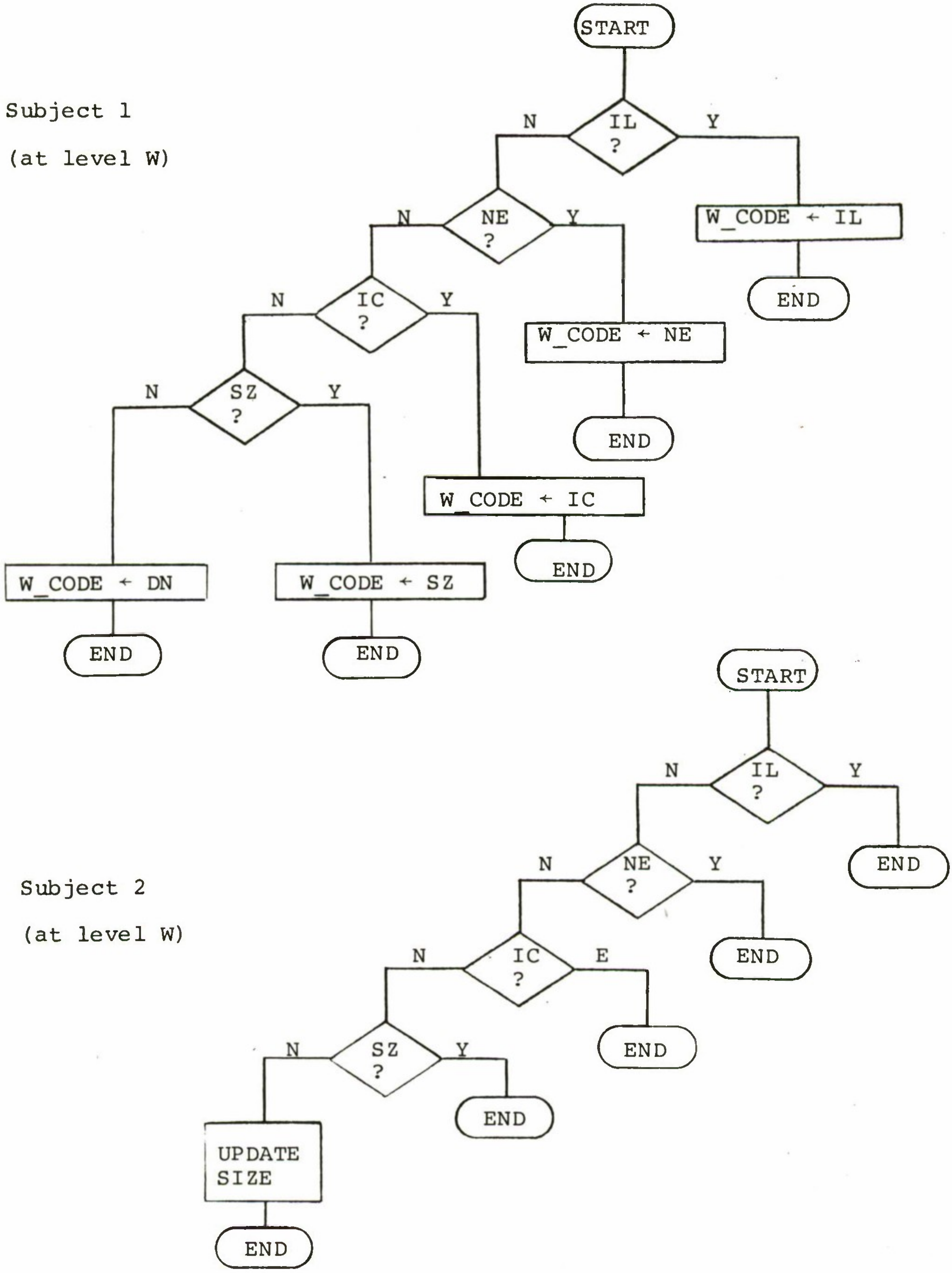
KDZ

Subject 1

(at level W)

Subject 2

(at level W)



PRIMITIVE: KDZ CASE: 1 SUBJECT: 1

CONDITIONS: IL
Accumulator level is not equal to
the user's current level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
n,t		W_CODE	W
CONSTANTS:	Unclass		
IL			
VARIABLES:			
K_LACC	W		
K_CUR_LEVEL	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KDZ CASE: 2 SUBJECT: 1

CONDITIONS: (~IL) ∧ NE
Non-existent object.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
n,t		W_CODE	W
CONSTANTS:	Unclass		
Ø,NE			
VARIABLES:			
K_LACC	LA		
K_CUR_LEVEL	W		
D_Z($\overline{id}$)	W		
K_CUR_ID	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL

PRIMITIVE: KDZ CASE: 3 SUBJECT: 1

CONDITIONS: (~IL) ∧ (~NE) ∧ IC
The accumulator does not contain
the required exact size component.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n,t	W	W_CODE	W
CONSTANTS: Ø,'Z',IC	Unclass		
VARIABLES: K_LACC K_CUR_LEVEL D_Z(id) K_IACC K_CUR_ID	 LA W W LA W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL

PRIMITIVE: KDZ CASE: 4 SUBJECT: 1

CONDITIONS: (~IL) ∧ (~NE) ∧ (~IC) ∧ SZ
The current exact size exceeds
the proposed maximum.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n,t	W	W_CODE	W
CONSTANTS: Ø,'Z',SZ	Unclass		
VARIABLES: K_LACC K_CUR_ID,K_CUR_LEVEL D_Z(id) K_IACC,K_VACC D_E(id) K_CUR_QTA	W W W LA W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL

PRIMITIVE: KDZ CASE: 5 SUBJECT: 1

CONDITIONS: (~IL) ∧ (~NE) ∧ (~IC) ∧ (~SZ)
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
n,t		W_CODE	W
CONSTANTS:	Unclass		
Ø,'Z',DN			
VARIABLES:			
K_LACC	W		
K_CUR_LEVEL,K_CUR_ID	W		
D_E(id),D_Z(id)	W		
K_CUR_QTA	W		
K_IACC,K_VACC	LA		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL

PRIMITIVE: KDZ CASE: 1 SUBJECT: 2

CONDITIONS: (~IL) ∧ (~NE) ∧ (~IC) ∧ (~SZ)
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
n,t		D_Z(id) K_CUR_QTA	W W
CONSTANTS:	Unclass		
Ø,'Z',DN			
VARIABLES: K_LACC	W		
K_CUR_LEVEL,K_CUR_ID	W		
D_E(id),D_Z(id)	W		
K_CUR_QTA	W		
K_IACC,K_VACC	LA		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: ~IL

A.1.3.12

(A) O-function KWA(n)

* Copy the accumulator contents to the working area *

parameter types

name n, contents (O,N,T,L,C)

exceptions

```
* Note that returning NOTHING instead of IL *
* transmits the same information            *
```

```
IL:  K_CUR_LEVEL ≯ K_LACC          * Current level must dominate    *
ND:  ~Discretionary_kwa            * No discretionary authorization *
```

effect

```
[1]  W_Fn   ← K_FACC               * Copy the format *
[2]  W_Vn   ← K_VACC               * Copy the values *
[3]  W_CODE ← DN
```

(B) Access table

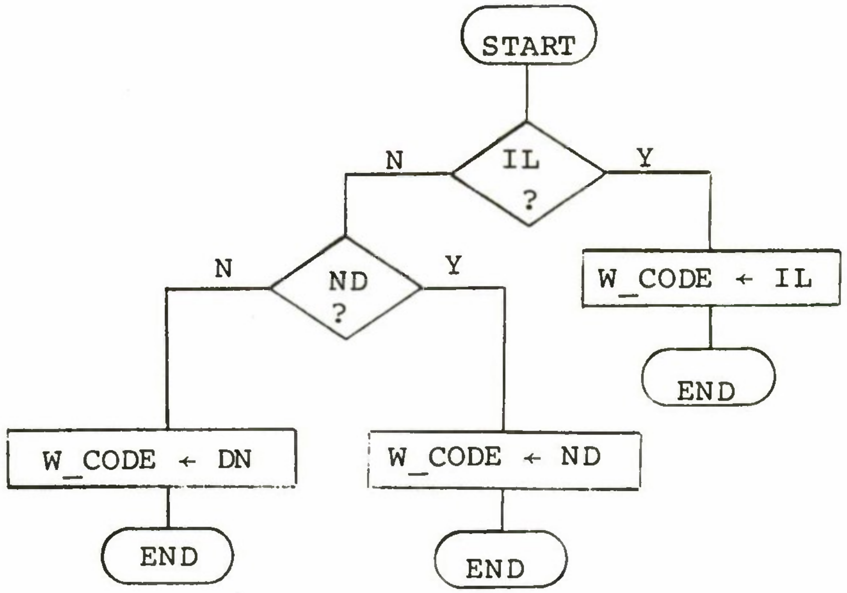
Variables Observed	Variables Modified	Variables Observed and Modified
K_CUR_LEVEL	W_Fn	
K_IACC,K_LACC	W_Vn	
K_CUR_ID	W_CODE	
K_OPEN		
K_FACC,K_VACC		

```
(C)  KWA  V-functions


(i)  V_function  Discretionary_kwa : boolean

          * Return a boolean indication of whether or not the   *
          * current user has discretionary access authorization *
          * to the accumulator contents.                        *

     range

          TRUE,FALSE

     derivation

          TRUE  IF  K_IACC[O] = K_CUR_ID               * DKD,DKQ and LIST_DOWN DATA *

     ELSE K_OPEN(K_IACC[O;N;T;L],RDSZ) IF  K_IACC[C] = 'E'   * Read current size?      *
     ELSE K_OPEN(K_IACC[O;N;T;L],RDHS) IF  K_IACC[C] = 'H'   * Read history?           *
     ELSE K_OPEN(K_IACC[O;N;T;L],RDPM) IF  K_IACC[C] = 'M'   * Read permission matrix? *
     ELSE K_OPEN(K_IACC[O;N;T;L],RSRV) IF  K_IACC[C] = 'R'   * Read reservation?       *
     ELSE K_OPEN(K_IACC[O;N;T;L],RETR) IF  K_IACC[C] = 'V'   * Read values?            *
     ELSE K_OPEN(K_IACC[O;N;T;L],RDSZ) IF  K_IACC[C] = 'Z'   * Read max. size?         *
     ELSE FALSE                        * Something must be wrong, fail.                *
```

KWA

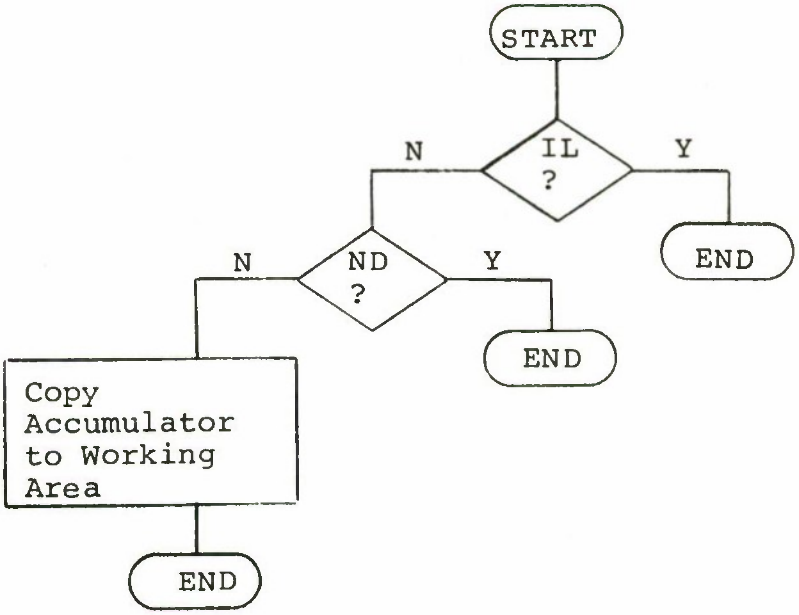
Subject 1

(at level W)



Subject 2

(at level W)



PRIMITIVE: KWA CASE: 1 SUBJECT: 1

CONDITIONS: IL
User's current level does not equal accumulator level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n	W	W_CODE	W
CONSTANTS: IL	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC	 W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: KWA CASE: 2 SUBJECT: 1

CONDITIONS: (~IL) ∧ ND
User has no discretionary authorization.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: n	W	W_CODE	W
CONSTANTS: RDSZ,RDHS,RDPM, RSRV,RETR,'E','H','M','R', 'V','Z',ND	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_CUR_ID K_OPEN[LA]	 W W W LA		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: Minimum K-level Invariant and ~IL

PRIMITIVE: KWA CASE: 3 SUBJECT: 1

CONDITIONS: (~IL) ∧ (~ND)
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
n		W_CODE	W
CONSTANTS: RDSZ,RDHS,RDPM,RSRV, RETR,'E','H','M','R','V','Z', DN	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_CUR_ID K_OPEN[LA] K_FACC K_VACC	W W W LA LA LA		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: Minimum K-level Invariant and ~IL

PRIMITIVE: KWA CASE: 1 SUBJECT: 2

CONDITIONS:

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
n		W_Fn W_Vn	W W
CONSTANTS: RDSZ,RDHS,RDPM,RSRV, RETR,'E','H','M','R','V','Z'	Unclass		
VARIABLES: K_CUR_LEVEL K_LACC K_CUR_ID K_OPEN K_VACC K_FACC	W W W LA LA LA		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

LEMMA: LA = W
PROOF: Minumum K-level Invariant and ~IL

A.1.4 SIGNON, SIGNOFF and MOVE

The security kernel functions, SIGNON, SIGNOFF and MOVE are discussed in Section V of this report. All three functions are in violation of the *-property for a successful execution, although the simple security principle is always upheld.

In this section, validations are included for these functions, to illustrate precisely how the *-property is violated. It will be seen that nothing other than a successful execution can induce a *-property violation and that all three functions may only be invoked by trusted subjects operating at the highest protection level.

The concept of the trusted subject, known as the User Controller Process (UCP), was introduced in the functional design report , Section 4.1. In initiating a sign-on to the DMS, the UCP must be restricted, by the hardware if necessary, to respond only to a human user's request to link to the DMS and not to the request of a process acting on behalf of a user.

The Data Base Administrator (DBA) is also considered to be a trusted subject and only a process acting at system high (SYS_HI), on behalf of the DBA, is permitted to declassify database objects. This is the purpose of the primitive MOVE.

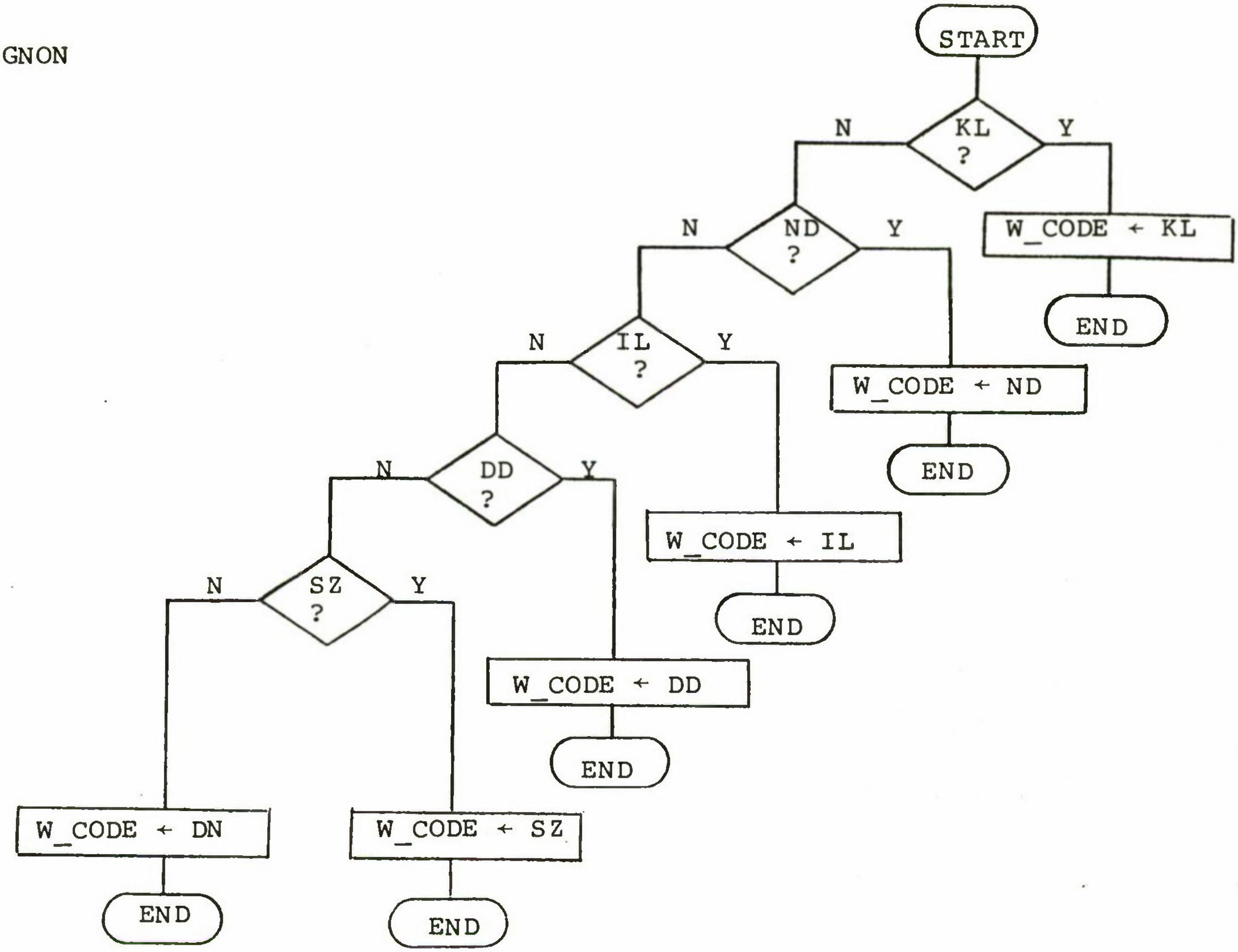
A.1.4.1 SIGNON

```
(A)  O-function  SIGNON(u,lv,s,v)

        * This is a request by the user control (and authentication) *
        * process to sign a user on to the secure DMS.  Users cannot *
        * request kernel functions until they have been signed on.   *
        * A working area and an initialized set of variables are     *
        * allocated to the user.  The user process is spawned.       *

   parameter types

        user u, level lv, size s, boolean v        * v is visibility indicator *

   abbreviations

        usent = (D_V(DBA,'DBA_ULIST','R',SYS_HI)  {D_F.USERID = u}  * relation type is user_ent *
                * Select the user's entry from the DBA's user relation. *
   exceptions

        KL:  (K_CUR_LEVEL ≠ SYS_HI) ∨ (K_CUR_ID ≠ UCP)     * It must be user control process *
        ND:   usent  = ∅                              * No such user                        *
        IL:  ⊢usent[MAX_LEVEL]⊣ ≯lv                   * Error if maximum level doesn't dominate *
        DD:  (u,*) ∈ ⊢D_Q(lv)⊣                        * User is signed on there already     *
        SZ:  s > ⊢usent[LIMIT]⊣ - ⊢usent[SUM]⊣        * Too much space requested            *

   effects
        For USER u:
        [1]  K_CUR_ID ← u                             * Set identification of K-Variables *
        [2]  K_CUR_LEVEL ← lv                         * Set the current level             *
        [3]  K_CUR_QTA ← s                            * Set session space quota           *
        [4]  ACTIVATE  K_CUR_TIME                     * Activate the timer for this user  *
        [5]  D_Q(lv) ← ⊢ ⊣ ∪ (u,v)                    * Append user entry to sign on list *
        [6]  usent[SUM] ← ⊢ ⊣ + s                     * Update user's sum of space used   *
        [7]  K_OPEN ← FALSE                           * Initialize whole 5 diménsional array *
        For UCP process:
        [8]  W_CODE ← DN                              * Set code for user control process *
```

(B) Access table	Variables Observed	Variables Modified	Variables Observed and Modified

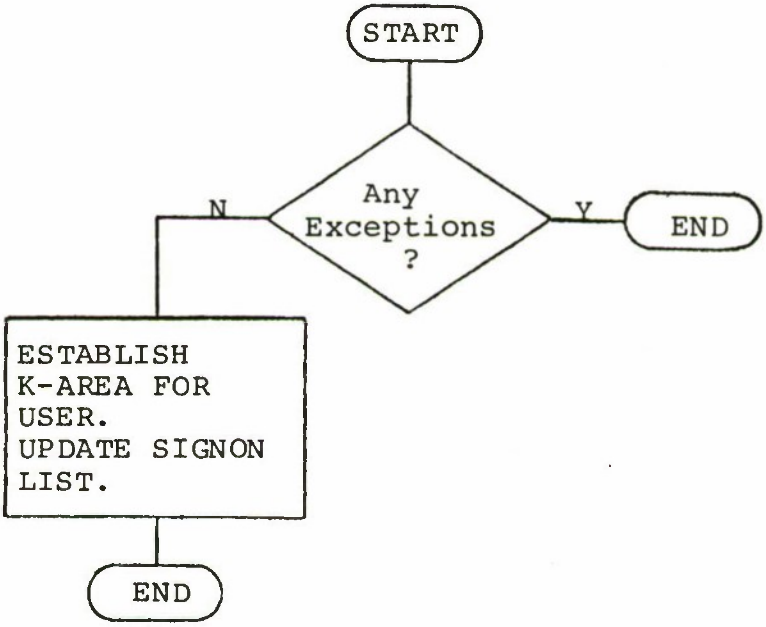
		K_CUR_ID,K_CUR_LEVEL
		K_CUR_QTA
		K_CUR_TIME
		K_OPEN
		W_CODE
			D_Q(lv)
			D_V(DBA,'DBA_ULIST')

SIGNON



Subject 2

(on behalf of UCP at level SYS_HI)



PRIMITIVE: SIGNON CASE: 1 SUBJECT: 1

CONDITIONS: KL
An attempt to invoke SIGNON by other than the trusted UCP at level SYS_HI.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: u,lv,s,r	W	W_CODE	W
CONSTANTS: UCP,SYS_HI,KL	Unclass		
VARIABLES: K_CUR_ID K_CUR_LEVEL	W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: SIGNON CASE: 2 SUBJECT: 1

CONDITIONS: (~KL) $\wedge$ ND
An attempt to sign-on by a non-registered data base user.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: u,lv,s,r	W	W_CODE	W
CONSTANTS: DBA,'DBA_ULIST','R', UCP,SYS_HI,∅,ND	Unclass		
VARIABLES: K_CUR_ID K_CUR_LEVEL usent	W W SYS_HI		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	W

LEMMA: W = SYS_HI
PROOF: ~KL

PRIMITIVE: SIGNON CASE: 3 SUBJECT: 1

CONDITIONS: (~KL) ∧ (~ND) ∧ IL
User's maximum level does not dominate
requested sign-on level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: u,lv,s,v	W	W_CODE	W
CONSTANTS: DBA,'DBA_ULIST','R', UCP,SYS_HI,Ø,MAX_LEVEL,IL	Unclass		
VARIABLES: K_CUR_ID K_CUR_LEVEL usent	 W W SYS_HI		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	W

LEMMA: W = SYS_HI
PROOF: ~KL

PRIMITIVE: SIGNON CASE: 4 SUBJECT: 1

CONDITIONS: (~KL) ∧ (~ND) ∧ (~IL) ∧ DD
User is already signed on at this level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: u,lv,s,v	W	W_CODE	W
CONSTANTS: DBA,'DBA_ULIST','R', UCP,SYS_HI,Ø,MAX_LEVEL,DD	Unclass		
VARIABLES: K_CUR_ID K_CUR_LEVEL usent D_Q(lv)	 W W SYS_HI lv		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	W

LEMMA: W = SYS_HI
PROOF: ~KL

PRIMITIVE: SIGNON CASE: 5 SUBJECT: 1

CONDITIONS: $(\sim KL) \wedge (\sim ND) \wedge (\sim IL) \wedge (\sim DD) \wedge SZ$
The user attempting to sign on
has requested too much space.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: u,lv,s,v	W	W_CODE	W
CONSTANTS: DBA,'DBA_ULIST,'R', UCP,SYS_HI,Ø,MAX_LEVEL,SZ	Unclass		
VARIABLES: K_CUR_ID K_CUR_LEVEL usent D_Q(lv)	 W W SYS_HI lv		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	W

LEMMA: W = SYS_HI
PROOF: ~KL

PRIMITIVE: SIGNON CASE: 6 SUBJECT: 1

CONDITIONS: $(\sim KL) \wedge (\sim ND) \wedge (\sim IL) \wedge (\sim DD) \wedge (\sim SZ)$
No exception.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: u,lv,s,v	W	W_CODE	W
CONSTANTS: DBA,'DBA_ULIST','R', UCP,SYS_HI,Ø,MAX_LEVEL,DN	Unclass		
VARIABLES: K_CUR_ID K_CUR_LEVEL usent D_Q(lv)	 W W SYS_HI lv		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	W

LEMMA: W = SYS_HI
PROOF: ~KL

PRIMITIVE: SIGNON CASE 1 SUBJECT: 2

CONDITIONS: (~KL) ∧ (~ND) ∧ (~IL) ∧ (~DD) ∧ (~SZ)
No exceptions. Establish the new DMS user.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
u,lv,s,v		K_CUR_ID(user)	lv
		K_CUR_LEVEL(user)	lv
CONSTANTS: DBA,'DBA_ULIST','R', SYS_HI,∅,MAX_LEVEL,SUM,DN, UCP,LIMIT	Unclass	K_CUR_QTA(user)	lv
		K_CUR_TIME(user)	lv
		D_Q(lv)	lv
VARIABLES:		usent[SUM]	SYS_HI
K_CUR_ID(UCP)	W		
K_CUR_LEVEL(UCP)	W		
usent	SYS_HI		
D_Q(lv)	lv		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	lv

The UCP must violate the *-property in order to sign users on to the system.

A.1.4.2 SIGNOFF

(A) SIGNOFF

```
     * The user control (and authentication) process will sign the current *
     * off the secure DMS, cleaning everything up.  (A possible means of   *
     * requesting a SIGNOFF is to turn the terminal off.)                  *

abbreviation

     usent    = (D_V(DBA,'DBA_ULIST','R',SYS_HI)  {D_F.USERID = K_CUR_ID}  * type is user_ent *
                * Select the current user's entry from the DBA's user relation. *

     open_obj = LOC_OP                          * This function locates all TRUE's in the   *
                                                * open table (K_OPEN), and returns 5-tuples *
                                                * consisting of their co-ordinates.         *
exceptions

     KL:  (K_CUR_LEVEL ≠ SYS_HI) v (K_CUR_ID ≠ UCP)     * It must be user control process *

effect
     For USER u:
     [1]  D_R(⊢K_RESERVE⊣) ← Ø                          * Remove all reservations      *
     [2]  D_O(⊢open_obj⊣) ← ⊢ ⊣ - ⊢K_CUR_ID⊣            * Remove all open list entries *
     [3]  D_Q(⊢K_CUR_LEVEL⊣) ← ⊢ ⊣ - (⊢K_CUR_ID⊣,*)*    Remove user entries            *
     [4]  usent[SUM] ← ⊢ ⊣ - ⊢K_CUR_QTA⊣                * Return unused space          *
     [5]  K_CUR_ID,K_CUR_LEVEL,K_CUR_QTA,K_CUR_TIME ← Ø    * Purge everything         *
     [6]  K_FACC,K_IACC,K_VACC,K_OPEN,K_RESERVE,K_LACC ← Ø
     [7]  K_FX,K_IX,K_VX,K_FY,K_IY,K_VY,K_FZ,K_IZ,K_VZ,K_LX,K_LY,K_LZ ← Ø

     For UCP Process:
     [8]  W_CODE ← DN
```

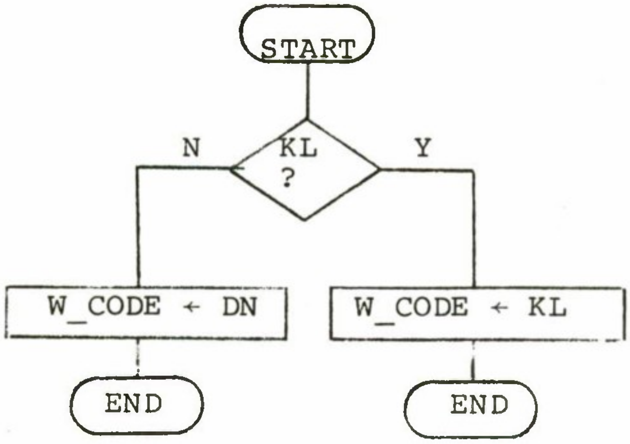
(B) Access table

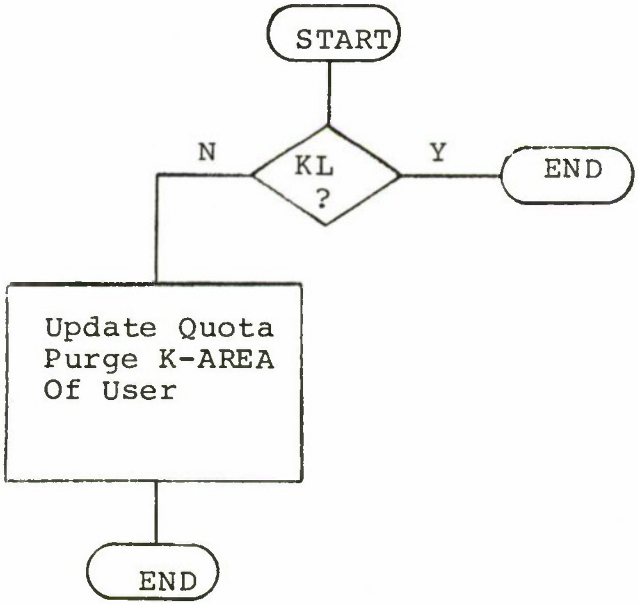
Variables Observed	Variables Modified	Variables Observed and Modified
	K_CUR_TIME	K_CUR_ID,K_CUR_LEVEL
	K_FACC,K_IACC	D_V(DBA,'DBA_ULIST')
	K_VACC,K_LACC	K_OPEN,K_RESERVE
	K_Fx,K_Ix,K_Vx	D_R(K_RESERVE)
	W_CODE	D_O(open_obj)
		D_Q(K_CUR_LEVEL)
		K_CUR_QTA

SIGNOFF

Subject 1

(at level W)





Subject 2

(for UCP at level SYS_HI)

PRIMITIVE: SIGNOFF CASE: 1 SUBJECT: 1

CONDITIONS: KL
An attempt to invoke SIGNOFF
by other than the UCP.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W	W_CODE	W
CONSTANTS: UCP,SYS_HI,KL	Unclass		
VARIABLES: K_CUR_LEVEL K_CUR_ID	W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: SIGNOFF CASE: 2 SUBJECT: 1

CONDITIONS: DN
No exceptions.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W	W_CODE	W
CONSTANTS: UCP,SYS_HI,DN	Unclass		
VARIABLES: K_CUR_LEVEL K_CUR_ID	W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: SIGNOFF CASE 1 SUBJECT: 2

CONDITIONS: No exceptions. The UCP executes SIGNOFF as the result of an explicit user request, or because of an implicit one, such as turning the terminal off.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
		D_R(⊢K_RESERVE⊣)	W
		D_O(⊢open_obj⊣)	W
CONSTANTS:	Unclass	D_Q(W)	W
UCP,SYS_HI,∅,DN		usent	SYS_HI
		K_CUR_ID	W
VARIABLES:		K_CUR_LEVEL	W
D_O(⊢open_obj⊣)	*[9]	K_CUR_QTA	W
K_CUR_ID	W	K_CUR_TIME	W
K_CUR_LEVEL	W	K_OPEN	*
D_Q(W)	W	ACC,X,Y,Z	*
usent	SYS_HI		
K_CUR_QTA	W		
K_OPEN	*		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	*[9]

[9] Any protection level

A.1.4.3 MOVE

(A) O-function MOVE(id,lv)

```
        * Move the specified object to the given protection level.  *
        * Only the DBA may use this primitive, since it potentially *
        * violates the star-property (i.e. modifies "lower" data.   *
```

parameter types

```
        identifier id (OWN,NAM,TYP,LEV), level lv, history (CREATION,USER,MODIFIED)
```

exceptions

```
        ND:  K_CUR_ID ≠ DBA                            * It must be the DBA              *
        IL:  K_CUR_LEVEL ≠ SYS_HI                      * User's max must dominate both *
        NE:  D_E(id) = ∅                               * Object doesn't exist.           *
        IR:  (OWN,NAM,TYP,ZERO) ∉ ⊢D_D(LEV)⊣          * Registrations are outstanding *
        DO:  D_O(id) ≠ ∅                               * Object is open (or reserved)   *
        DD:  (OWN,NAM,TYP,*) ∈ ⊢D_D(lv)⊣              * Name has already been used     *
```

effect

```
        [1]  D_D(lv) ← ⊢ ⊣ ∪ (OWN,NAM,TYP,ZERO)       * Append a new entry            *
        [2]  D_E(OWN,NAM,TYP,lv) ← ⊢D_E(id)⊣           * Copy all component entities *
        [3]  D_F(OWN,NAM,TYP,lv) ← ⊢D_F(id)⊣           * containing the object        *
        [4]  D_H(OWN,NAM,TYP,lv) ←(⊢D_H(id)[CREATION]⊣,'DBA',K_CUR_TIME)
                                        * Take creation date, create rest of history.  *
        [5]  D_M(OWN,NAM,TYP,lv) ← ⊢D_M(id)⊣
        [6]  D_V(OWN,NAM,TYP,lv) ← ⊢D_V(id)⊣
        [7]  D_Z(OWN,NAM,TYP,lv) ← ⊢D_Z(id)⊣
        [8]  D_E(id),D_F(id),D_H(id),D_M(id),D_V(id),D_Z(id) ← ∅   * Purge old object*
        [9]  D_D(LEV) ← ⊢ ⊣ − (OWN,NAM,TYP,ZERO)      * Remove the directory entry   *
        [10] W_CODE ← DN
```

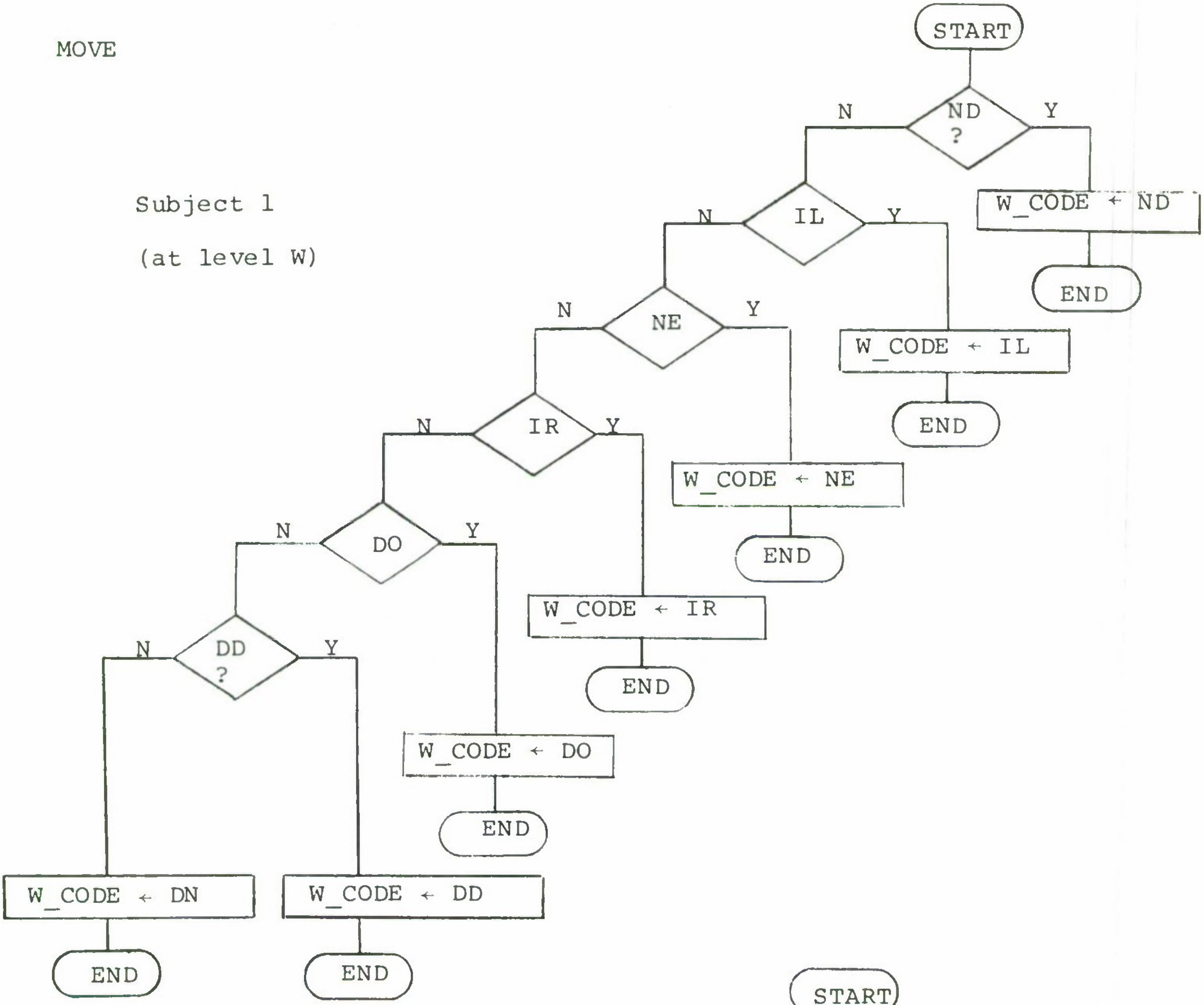
(B) Access table	Variables Observed	Variables Modified	Variables Observed and Modified

	K_CUR_ID,K_CUR_LEVEL
	D_V(DBA,'DBA_ULIST')
		W_CODE	D_D(lv)
			D_D(LEV)
			D_E(id),D_F(id)
			D_H(id),D_M(id)
			D_V(id),D_Z(id)

MOVE

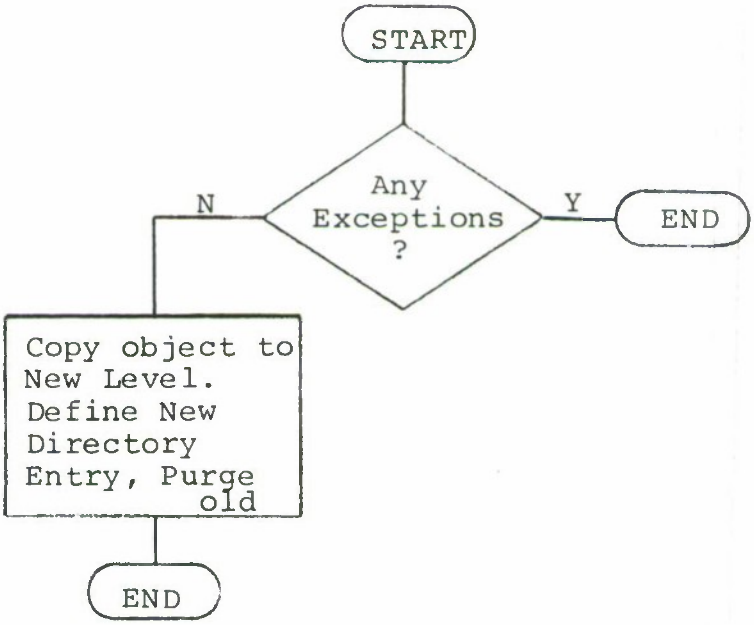
Subject 1

(at level W)



Subject 2

(at level SYS_HI)


START
Any Exceptions ?
N
Y
END
Copy object to New Level. Define New Directory Entry, Purge old
END

PRIMITIVE: MOVE CASE: 1 SUBJECT: 1

CONDITIONS: ND
User is not the DBA.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id,lv	W	W_CODE	W
CONSTANTS: DBA,ND	Unclass		
VARIABLES: K_CUR_ID	W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: MOVE CASE: 2 SUBJECT: 1

CONDITIONS: (~ND) ∧ IL
DBA is not signed on at the system high level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id,lv	W	W_CODE	W
CONSTANTS: DBA,SYS_HI,IL	Unclass		
VARIABLES: K_CUR_ID K_CUR_LEVEL	 W W		
HIGHEST LEVEL OBSERVED:	W	LOWEST LEVEL MODIFIED:	W

PRIMITIVE: MOVE CASE: 3 SUBJECT: 1

CONDITIONS: (~ND) ∧ (~IL) ∧ NE
Object does not exist.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
id,lv		W_CODE	SYS_HI
CONSTANTS:	Unclass		
DBA,SYS_HI,Ø,NE			
VARIABLES:			
K_CUR_ID	SYS_HI		
K_CUR_LEVEL	SYS_HI		
D_E(id)	lv		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	SYS_HI

PRIMITIVE: MOVE CASE: 4 SUBJECT: 1

CONDITIONS: (~ND) ∧ (~IL) ∧ (~NE) ∧ IR
There are outstanding registrations.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	W		
id,lv		W_CODE	SYS_HI
CONSTANTS:	Unclass		
DBA,SYS_HI,Ø,ZERO,IR			
VARIABLES:			
K_CUR_ID	SYS_HI		
K_CUR_LEVEL	SYS_HI		
D_E(id)	lv		
D_D(lv)	lv		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	SYS_HI

PRIMITIVE: MOVE CASE: 5 SUBJECT: 1

CONDITIONS: (~ND) ∧ (~IL) ∧ (~NE) ∧ (~IR) ∧ DO
Object is open at present.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id,lv	W	W_CODE	SYS_HI
CONSTANTS: DBA_SYS_HI,Ø,ZERO,DO	Unclass		

VARIABLES:
K_CUR_ID
K_CUR_LEVEL
D_E(id)
D_D(lv)
D_O(id)
	
	SYS_HI
	SYS_HI
	lv
	lv
	lv		

HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	SYS_HI

PRIMITIVE: MOVE CASE: 6 SUBJECT: 1

CONDITIONS: (~ND) ∧ (~IL) ∧ (~NE) ∧ (~IR) ∧ (~DO) ∧ DD
The object already exists at the new level.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id,lv	W	W_CODE	SYS_HI
CONSTANTS: DBA,SYS_HI,Ø,ZERO,DD	Unclass		

VARIABLES:
K_CUR_ID
K_CUR_LEVEL
D_E(id)
D_D(lv)
D_O(id)
	
	SYS_HI
	SYS_HI
	lv
	lv
	lv		

HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	SYS_HI

PRIMITIVE: MOVE CASE 7 SUBJECT: 1

CONDITIONS: (~ND) ∧ (~IL) ∧ (~NE) ∧ (~IR) ∧ (~DO) ∧ (~DD)
No exceptions.

	OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS:	id,lv	W	W_CODE	SYS_HI
CONSTANTS:	DBA,SYS_HI,Ø,ZERO,DN	Unclass		
VARIABLES:				
	K_CUR_ID	SYS_HI		
	K_CUR_LEVEL	SYS_HI		
	D_E(id)	lv		
	D_D(lv)	lv		
	D_O(id)	lv		
	D_F(id)	lv		
	D_H(id)	lv		
	D_M(id)	lv		
	D_V(id)	lv		
	D_Z(id)	lv		
	K_CUR_TIME	SYS_HI		
HIGHEST LEVEL OBSERVED:		SYS_HI	LOWEST LEVEL MODIFIED:	SYS_HI

PRIMITIVE: MOVE CASE 1 SUBJECT: 2

CONDITIONS: (~ND) ∧ (~IL) ∧ (~NE) ∧ (~IR) ∧ (~DO) ∧ (~DD)
No exceptions. Copy object to new level; append entry to new directory and delete entry from old.

OBSERVED	LEVEL	MODIFIED	LEVEL
PARAMETERS: id,lv	W	D_D(lv)	lv
		D_D(new-lv)	lv
		D_E(id)	lv
CONSTANTS: DBA,SYS_HI,∅,ZERO,DN	Unclass	D_F(id)	lv
		D_H(id)	lv
		D_M(id)	lv
VARIABLES:		D_V(id)	lv
K_CUR_ID	SYS_HI	D_Z(id)	lv
K_CUR_LEVEL	SYS_HI	D_E(new-id)	lv
D_E(id)	lv	D_F(new-id)	lv
D_D(lv)	lv	D_H(new-id)	lv
D_O(id)	lv	D_M(new-id)	lv
D_F(id)	lv	D_V(new-id)	lv
D_H(id)	lv	D_Z(new-id)	lv
D_M(id)	lv		
D_V(id)	lv		
D_Z(id)	lv		
K_CUR_TIME	SYS_HI		
HIGHEST LEVEL OBSERVED:	SYS_HI	LOWEST LEVEL MODIFIED:	lv

This function violates the *-property in a controlled manner since only the DBA may reclassify information. It does not violate the SS-property because the DBA must have a system high current level.

APPENDIX II

RESULTS OF THE VALIDATION

The course of the validation revealed certain problems in the specification. These problems and their solution are summarized in table A.2.1.

Table A.2.1 Problems with the Specifications

	Problem Description	Problem Type	Solution	Functions Affected
1.	*'-property violations	Design error	Define new entities: K_LACC,K_LX,K_LY,K_LZ at level = K_CUR_LEVEL	DKD,DKE,DKH,DKM, DKQ,DKR,DKV,DKZ, LIST_DOWN,ASSIGN, KWA
2.	Accumulator could not be "lower" than parameter data	Design error	Set minimum kernel level = K_CUR_LEVEL	Same as 1.
3.	"NO" exceptions for "higher" data provided a communication path down	Design error	Purge accumulator if exception is TRUE	DKE,DKH,DKM, DKR,DKV,DKZ
4.	Using quota at "higher" levels constituted a "write-down" to K_CUR_QTA	Design error	Restrict K_CUR_QTA modification to current level.	INIT,KDZ
5.	Failed to check for a string (i.e. single tuple). Also, K_CUR_ID,K_CUR_LEVEL omitted from access table	Simple omission	Include them	WKB
6.	Return code condition and asterisks on exception codes were omitted	Simple omission	Include them	INIT
7.	ND exception was redundant (already in Unique_keys of IV exception)	Logic error	Remove ND	APPEND,Unique_keys

(Table A.2.1 continued...)

	Problem Description	Problem Type	Solution	Functions Affected
8.	Exceptions IT and IL were in the wrong order	Logic error	Switch them	APPEND
9.	There was no check that SIGNON and SIGNOFF were executed by UCP.	Simple omission	Include check	SIGNON,SIGNOFF
10.	Superfluous format check	Logic error	Remove it	KDZ
11.	Return code condition excluded a check for kernel to kernel transfer.	Serious omission	Include it	ASSIGN
	Asterisk on W_CODE was omitted	Simple omission	Include it	
12.	Incorrect (obsolete) OPEN ARRAY test for "NO" exception.	Oversight	Replace in specs: *NO: K_OPEN(id,EXPM)=FALSE *NO: K_OPEN(id,STOR)=FALSE	 KDM KDV,WDV

REFERENCES

[1] G. Kirkby and M. Grohn, On Specifying the Functional Design for a Protected DMS Tool, ESD-TR-77-140, I.P. Sharp Associates Limited, Ottawa, Canada, March 1977.

[2] M. Grohn, A Model of a Protected Data Management System, ESD-TR-76-289, I.P. Sharp Associates Limited, Ottawa, Canada, June 1976.

[3] J.K. Millen, Security Kernel Validation in Practice, Communications of the ACM, Volume 19, Number 5, May 1976, pp. 243-250.

MISSION
OF THE
DIRECTORATE OF COMPUTER SYSTEMS ENGINEERING

The Directorate of Computer Systems Engineering provides ESD with technical services on matters involving computer technology to help ESD system development and acquisition offices exploit computer technology through engineering application to enhance Air Force systems and to develop guidance to minimize R&D and investment costs in the application of computer technology.

The Directorate of Computer Systems Engineering also supports AFSC to insure the transfer of computer technology and information throughout the Command, including maintaining an overview of all matters pertaining to the development, acquisition, and use of computer resources in systems in all Divisions, Centers and Laboratories and providing AFSC with a corporate memory for all problems/solutions and developing recommendations for RDT&E programs and changes in management policies to insure such problems do not reoccur.